UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

ESTUDIO DE ALGORITMOS DE CONTROL DE TRAYECTORIA CON ROS PARA UN HELICÓPTERO COAXIAL

TRABAJO FINAL DE MÁSTER

Máster Universitario en Ingeniería de Sistemas Automáticos y Electrónica Industrial.

Autor: Carlos Eduardo Barrionuevo Sánchez

Tutores: Bernardo Morcego Seix.

Rámon Perez

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Agradecimiento

Quiero expresar mis sinceros agradecimientos a mis tutores Bernardo Morcego y Ramón Pérez, dos profesionales excelentes, quienes me han guiado a través de este proceso, con ideas, consejos y enseñanzas.

A Marcela Venegas, por acompañarme en todo este tiempo, y ayudarme a seguir adelante.

A mi familia por todo el esfuerzo que han hecho para darme esta oportunidad de seguir avanzando en mi camino profesional.

1. INTRODUCCIÓN

1.1 Justificación

Dentro del campo de la robótica las aeronaves no tripuladas, UAV por sus siglas en inglés, han extendido su uso y desarrollo en diferentes áreas como las de: Reconocimiento aéreo, asistencia en rescates, filmaciones cinematográficas, transporte de carga ligera y otras. Esto ha sido posible gracias a un desarrollo conjunto de diferentes ramas entre las que se encuentran los sistemas de control de vuelo, los cuales son parte fundamental del UAV ya que le permiten solventar diferentes perturbaciones tanto internas como externas.

Por otro lado, la visión por computadora o visión artificial, es también parte de la robótica que se ha desarrollado notablemente en los últimos años, y que permite obtener una gran cantidad de información del entorno, sustituyendo así la aplicación de una gran cantidad de sensores de diferentes tipos, lo cual conlleva un ahorro tanto de costos como de instalación y mantenimiento. Es por esto que se puede utilizar conjuntamente con los sistemas de control de vuelo para proveer toda la información de ubicación y orientación del UAV, necesaria para que éste pueda maniobrar adecuadamente.

1.2 Objetivo

El objetivo del trabajo es adaptar un sistema de visión artificial, diseñado en la plataforma ROS, para que calcule correctamente la posición y orientación de una estructura móvil, con cuatro marcadores, y validar esta información para enviarla a un ordenador con Matlab y generar ordenes de control necesarias para mantener un helicóptero coaxial en un punto de equilibrio.

A partir de este objetivo principal se plantean los siguientes objetivos específicos:

- Aprender sobre la plataforma ROS y como se utiliza en el desarrollo de aplicaciones complejas
- Mejorar y verificar la robustez del sistema de visión artificial.
- Establecer una comunicación adecuada entre la información de las cámaras y el controlador.
- Incorporar al helicóptero y al modelo los marcadores necesarios para el funcionamiento del sistema de visión artificial

- Implementar algún controlador previo del helicóptero y realizar los ajustes necesarios para su puesta en marcha
- Comprobar el funcionamiento del sistema ante pequeñas perturbaciones

1.3 Alcance

Para cumplir con los objetivos anteriores es necesario:

- Entender el funcionamiento de la plataforma ROS
- Entender el funcionamiento de los helicópteros coaxiales
- Fijar los marcadores de posición desarrollados al helicóptero
- Verificar el funcionamiento del sistema de visión de artificial
- Mejorar el sistema de visión artificial
- Desarrollar el enlace entre el sistema de visión artificial y el sistema de control
- Realizar pruebas del sistema de control
- Modificaciones y mejoras necesarias
- Detallar el trabajo realizado en la memoria del TFM

Cabe destacar que el presente trabajo parte de trabajos previos en los que se ha modelado el helicóptero, se ha creado un bloque con el modelo para simular su comportamiento en Simulink y se ha diseñado un controlador que funcionó con otro tipo de cámaras. Así como también se ha diseñado el programa de visión artificial y se comprobado la precisión de sus cálculos en posición y orientación.

1.4 Especificaciones

Para el desarrollo del presente trabajo se necesita hardware y software específicos, así como también una serie de condiciones que se van a detallar a continuación.

Como parte del hardware, se va a utilizar un helicóptero Walkera 5#10, 4 Cámaras infrarrojas, 5 ordenadores y una estructura de marcadores para el sistema de visión artificial.

Como parte del software se va utilizar: 4 Ordenadores con sistema Operativo Ubuntu con el software OpenCV y ROS. 1 ordenador con sistema Operativo Windows 10 con software Matlab 2016. Las versiones y condiciones del software se explican en detalle más adelante.

Finalmente, como parte de las condiciones físicas se establece un área de vuelo de 2m x 2m x 2m, como mínimo.

1.5 Antecedentes

El trabajo con aeronaves no tripuladas pequeñas (UAVs) y el diseño y pruebas de diferentes estrategias y algoritmos de control para las mismas comenzó en septiembre de 2007 dentro del área de investigación del departamento de Ingeniería de Sistemas, Automática e Informática Industrial (ESAII). Parte del interés por realizar este trabajo surge de que estos sistemas tienen gran dinámica y por tanto son difíciles de controlar, ya que en general se trata de sistemas multivariables, no lineales, inestables y que presentan un gran acoplo entre sus variables.

Para poder solventar estos problemas, ha sido necesario identificar la estabilidad de los diferentes sistemas, identificar donde sea posible trabajar en un lazo cerrado, localizar un objeto mediante visión artificial y determinar su orientación aun cuando se encuentra en movimiento. Además, que se plantea la necesidad de diseñar estructuras de auto pilotaje, y de crear algoritmos adaptativos que permitan cambiar el modo de vuelo.

Los líderes de este proyecto son Bernardo Morcego, Ramón Pérez y Josep Cugeró, quienes han trabajado con los estudiantes: Daniele Laccetti, Gabriel Portell, Alex Vargas, Sebastià Roca, David Heredia, Rubèn Acedo, Albert Jordà, Gerard Graugés, Tomeu Rubí.

Ellos han dedicado un gran esfuerzo en determinar la posición y altitud del helicóptero con una buena precisión, mediante diferentes paquetes en ROS. Utilizando diferentes librerías y el conocimiento físico de los marcadores añadidos, han sido capaces de proporcionar las coordenadas físicas del helicóptero, así como también otras variables necesarias para el control, como es la orientación a lo largo de los diferentes ejes espaciales. Todo esto trabajando sobre un helicóptero de bajo costo, y sin añadir ningún sensor adicional al modelo.

Sin embargo, estos programas han sido realizados en versiones de software anteriores a la más recientes, las cuales, si bien aún se continúan utilizando ampliamente, a futuro representaran un problema ya que no podrán utilizar herramientas más modernas que simplifiquen el trabajo y permiten un mejor desarrollo del proyecto.

1.6 Presupuesto

Los elementos necesarios para realizar este trabajo son de propiedad del laboratorio de control avanzado de la UPC, quienes han trabajado en proyectos anteriores con estos materiales, además que los ordenadores cumplen otras funciones y tienen conectadas diferentes maquetas del laboratorio. Por este motivo no se ha tomado en cuenta los costos de estos elementos, sino que el presupuesto necesario dependerá de las horas de trabajo necesarias para su desarrollo. Una aproximación de las horas empleadas en el TFM se puede ver en la tabla 1.

Tabla 1 Horas de Trabajo

CONCEPTO	Nº DE HORAS
Software Necesario	
Pruebas Conectividad	10
Actualización de Software versión más reciente	30
Pruebas	10
Sistema de Visión Artificial	
Pruebas del sistema de visión previo	30
Análisis de los resultados	10
Cambios en el algoritmo	50
Helicóptero Coaxial	
Pruebas de los materiales existentes	10
Cambios estructura física	30
Pruebas	10
Mejoras	20
Total de Horas	210
Total Euros	2100 €

1.7 Planificación

El tiempo de ejecución del presente trabajo se ha dividido en meses de trabajo y la organización del mismo se observa en la Tabla 2.

Tabla 2 Cronograma

Actividades	MES					
	Marzo	Abril	Mayo	Junio	Julio	Septiembre
Redacción Memoria						
Aprendizaje ROS						
Comunicación entre elementos						
Pruebas Visión Artificial						
Calibración Cámaras						
Modificaciones Sistema de Visión Artificial						
Modificaciones Físicas						
Implementación Control						
Pruebas de Control						

2 SISTEMAS OPERATIVOS Y SOFTWARE

En este capítulo se explicará los diferentes programas utilizados en trabajos previos, y después el cómo se ha realizado la migración a la versión más actual de dichos programas. Específicamente se hablará del trabajo “Diseño de un sistema de detección de marcadores con ROS y visión por computadora”, efectuado por el estudiante Cristian Martínez Céspedes, y del trabajo “Estudio para el modelado y control de un helicóptero coaxial, Unmaned Aerial Vehicle (UAV)” desarrollado por Tomeu Rubí Perelló.

2.1 Ubuntu

Ubuntu es un sistema operativo tipo Linux diseñado para ordenadores, *tablets* y *smartphones*, el cual está basado en Debian y utiliza Ubuntu Touch como editor. Este sistema operativo es ampliamente utilizado por desarrolladores ya que funciona sobre código abierto.

El sistema de visión artificial se ha desarrollado sobre la plataforma ROS, la cual está orientada para trabajar sobre sistema operativo UNIX, es decir Ubuntu Linux. Si bien existe una versión de ROS para Windows esta es considerada como “experimental” y no posee toda la documentación de respaldo que existe en Ubuntu. Por tal motivo, si se cuenta con un ordenador que trabaje con Windows, existen 2 opciones de trabajo. La primera consiste en crear una o varias particiones del disco duro, para poder instalar el sistema operativo Ubuntu en alguna de ellas. Esta opción es la más recomendable ya que se consigue aprovechar al máximo el hardware del ordenador.

La siguiente opción si no se desea realizar una partición del disco duro, consiste en utilizar máquinas virtuales. Esta opción se puede realizar mediante diferentes programas como “Oracle VM Virtual Box” o “VMware player” entre otros muchos programas para crear máquinas virtuales. La ventaja que tiene esta opción es que, en el caso de usuarios inexpertos, se evita el riesgo de perder toda la información al realizar una partición de disco de forma errónea, y se mantiene el sistema operativo original con todos sus archivos y programas intactos.

La desventaja se encuentra también en este hecho ya que, al trabajar sobre el sistema operativo original, este se encuentra funcionando de fondo todo el tiempo, y consumo recursos de memoria, cache y demás, lo cual puede entorpecer la máquina virtual.

Otra de las desventajas de trabajar con máquina virtual es que se necesita cierto conocimiento para configurar las comunicaciones, ya que en la mayoría de casos es necesario realizar puentes virtuales entre un sistema operativo y otro, para que ambos puedan acceder sin conflicto a los puertos de hardware y demás.

Los proyectos se habían estado trabajando en la versión 14 de Ubuntu, pero actualmente ya existe la versión 16 e incluso la versión 17. Debido a que la versión 17 ha llegado hace poco tiempo, se decidió actualizar el sistema operativo a la versión 16, la cual es la versión LTS, es decir “Long Term Support”, que incluye actualizaciones para nuevo hardware, parches de seguridad y actualizaciones de “Ubuntu Stack” es decir la infraestructura computacional en la nube, además de que es una versión muy estable y no presentar fallos significativos.

Esta actualización se realizó en particiones del disco de los ordenadores del laboratorio de control avanzado, creando primero un ordenador con todos los diferentes sistemas operativos y sus respectivos programas para trabajar, para después crear una copia de este único equipo e instalarla en los demás ordenadores.

Dentro de este proyecto se va a utilizar ampliamente la llamada terminal de Ubuntu, que es un intérprete de comandos. Esto es un programa que toma la entrada del usuario, por ejemplo, las órdenes que teclea, y las traduce a instrucciones de máquina. En cualquier GNU/Linux existe una terminal o consola que abre un shell o intérprete de comandos.

2.2 OpenCv

OpenCv (Open source Computer Vision Library) es una librería de software que trabaja sobre código abierto, para desarrollar visión por computadora y aprendizaje. Fue construida para proveer una infraestructura común para el desarrollo de aplicaciones de visión artificial, así como para acelerar el uso de la llamada “precepción de máquina” en productos comerciales. (Team_OpenCV)

Anteriormente se trabajaba con la versión 2.4 de este software, sin embargo, actualmente se cuenta con versiones de OpenCV más actuales, e incluso se ha desarrollado una 3era versión del software, la cual mejora la velocidad del mismo y dentro de la cual se encuentran las últimas

actualizaciones, convirtiéndose en la última versión estable del programa. Por este motivo se actualizó el programa de la versión 2.4 a la versión 3.0.

Estas versiones presentan diferencias en los diseños de encabezados, cambios a nivel de algoritmos, implementación de módulos de aprendizaje de la máquina y más actualizaciones, desarrolladas para la última versión, que deben ser tomadas en cuenta al momento de realizar la migración del código de una versión a otra. En nuestro caso esta migración afecta a 2 programas ejecutables, que se utilizan para la visión artificial.

El primero es el programa “proba_estructura” el cual crea una ventana en donde se observa las imágenes que recibe la cámara, efectúa la detección de los marcadores de acuerdo a las condiciones almacenadas en un puntero, para luego mostrarlos junto con la imagen y finalmente imprimir esa información en pantalla. El segundo programa se llama camera1, y es el programa que se implementa en cada uno de los ordenadores, cambiando el número de cámara al correspondiente y es el encargado de realizar todo el sistema de visión artificial.

En estos programas se han realizado los siguientes cambios:

A.

Versión 2.4: SimpleBlobDetector detector(params);

Versión 3.0: Ptr<SimpleBlobDetector> detector= SimpleBlobDetector::create(params);

Aquí se crea un objeto llamado detector, que es del tipo puntero, el cual apunta al método de crear un detector de blobs simple, con la información almacenada en “params”.

B.

Versión 2.4: detector.detect(imatge, keypoints);

Versión 3.0: detector->detect(imatge, keypoints);

Aquí se llama al objeto llamado detector y se invoca el método propio de su clase llamado “detect” el cual se encarga de encontrar los blobs en la imagen que recibe en “imatge” y almacenar dicha información en el vector “keypoints”.

2.3 Robotic Operating System (ROS)

ROS es un sistema operativo de código abierto y gratuito, desarrollado para facilitar la programación de Robots, ya que contiene diferentes aplicaciones desarrolladas que permiten realizar el control de dispositivos a bajo nivel, así como también gestionar el envío de información entre diferentes procesos. Esto representa una gran ayuda al momento de trabajar con pequeños robots, los cuales no poseen grandes recursos a nivel de software, pero sí todas las funcionalidades necesarias a nivel de hardware.

Estas aplicaciones están desarrolladas para gestionar diferentes equipos los cuales pueden o no estar ubicados en diferentes ordenadores y trabajar en conjunto de manera ordenada. Otra de las ventajas de ROS es que permite la creación y compilación de nuevo código de acuerdo a las necesidades y capacidades del usuario.

Dentro de su estructura de funcionamiento ROS distingue 2 formas de comunicación diferentes, las cuales pueden ser mediante un modelo de publicador y subscriptor, en el cual diferentes publicadores envían constantemente información a la red, y cada subscriptor en la red se configura para estar suscrito a algún tópico y así saber de dónde debe tomar dicha información.

El proyecto de visión artificial por computadora fue desarrollado en la versión de ROS “Jade Turtle” que correspondería a la versión 10 de ROS, sin embargo, ahora se cuenta ya con la versión 11 que se denomina “Kinetic Kame”. Debido al cambio de versión de ROS los programas ejecutables creados anteriormente no se pueden utilizar, ya que los archivos de configuración necesarios para el ejecutable son totalmente diferentes entre dichas versiones.

Por tal motivo es necesario partir del código del programa que se encuentra desarrollado en C++ y volver a crear los paquetes y ejecutables de ROS para la versión “kinetic”.

2.3.1 Verificar Instalación

Una vez que se ha instalado la nueva versión de ROS, o que se ha instalado por primera vez, dependiendo del ordenador de trabajo, es necesario comprobar que ésta haya sido exitosa. Se puede emplear el siguiente comando:

$ env | grep ROS

Este comando se utiliza para enlistar todas las variables ambientales de determinado programa, en este caso ROS. Aquí vamos a observar la siguiente información importante

- La raíz en donde se encuentran los archivos de configuración
- La dirección donde se almacenan los paquetes de ROS
- La dirección del nodo máster en la red (ROS_MASTER_URI)
- Los directorios LISP. en donde LISP es una biblioteca de cliente para escribir nodos ROS en idiomática Common Lisp. La librería se escribe para facilidad de uso, el scripting rápido de nodos y la depuración interactiva de un sistema ROS en ejecución.
- La versión actual de ROS
- La dirección de los directorios “etc”

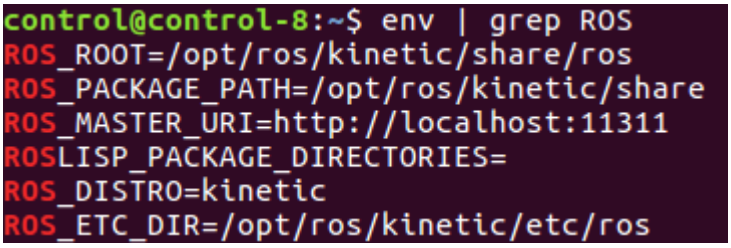

Figura 1 Resultado Variables de Configuración

Cuando se ha realizado una instalación correcta se debe observar un resultado similar al de la figura 1.

2.3.2 Espacio de Trabajo (workspace)

Una vez que se han verificado las variables de ambiente de ROS, resulta muy útil y recomendable trabajar a través de espacios de trabajo. Estos espacios de trabajo permiten tener almacenada y ordenada, toda la información necesaria para compilar un programa y crear su ejecutable en ROS.

El espacio de trabajo utilizado en este proyecto se llama “Catkin Tools” y fue diseñado originalmente para construir eficientemente numerosos proyectos de CMake interdependientes, pero desarrollados por separado. Esto fue desarrollado por la comunidad de Robot Operating System (ROS), originalmente como un sucesor de la meta estándar “buildtool rosbuild”.
Para instalarlo se utiliza el comando
$ sudo apt-get install python-catkin-tools

Y se debe aceptar la instalación del mismo. Una vez que se haya instalado, no es necesario reiniciar el equipo, y se puede utilizar inmediatamente para compilar un proyecto. Primero se debe ejecutar el contenido del script “setup.bash”, que como su nombre indica contiene la información de la configuración, en el shell actual, y por tanto con las variables de entorno del shell actual. El código es el siguiente:

$ source /opt/ros/kinetic/setup.bash

$ mkdir -p ~/carlos/src

$ cd ~/carlos

$ catkin_make

Una vez ejecutados estos comandos se encuentra el siguiente resultado:

```
control@control-8:~/carlos$ catkin_make
Base path: /home/control/carlos
Source space: /home/control/carlos/src
Build space: /home/control/carlos/build
Devel space: /home/control/carlos/devel
Install space: /home/control/carlos/install
Creating symlink "/home/control/carlos/src/CMakeLists.txt" pointing to
 "/opt/ros/kinetic/share/catkin/cmake/toplevel.cmake"
####
#### Running command: "cmake /home/control/carlos/src -DCATKIN_DEVEL_P
REFIX=/home/control/carlos/devel -DCMAKE_INSTALL_PREFIX=/home/control/
carlos/install -G Unix Makefiles" in "/home/control/carlos/build"
####
-- The C compiler identification is GNU 5.4.0
-- The CXX compiler identification is GNU 5.4.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Using CATKIN_DEVEL_PREFIX: /home/control/carlos/devel
-- Using CMAKE_PREFIX_PATH: /opt/ros/kinetic
-- This workspace overlays: /opt/ros/kinetic
-- Found PythonInterp: /usr/bin/python (found version "2.7.12")
-- Using PYTHON_EXECUTABLE: /usr/bin/python
-- Using Debian Python package layout
-- Using empy: /usr/bin/empy
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CATKIN_TEST_RESULTS_DIR: /home/control/carlos/build/test_resu
lts
```

Figura 2 Resultado al compilar un proyecto

En la figura 2 podemos observar como el comando “catkin_make” va a definir unos encabezados para que el programa sepa de donde obtiene la información y donde la va a almacenar. Una vez que ha terminado exitosamente vamos a observar que dentro de la carpeta Carlos, que es el nombre del espacio de trabajo que hemos creado, se han creado subcarpetas llamadas “src”, “devel” y “build”, que es donde se irá almacenando y organizando toda la información necesaria para crear un paquete de ROS. Esto se puede observar en la figura 3.

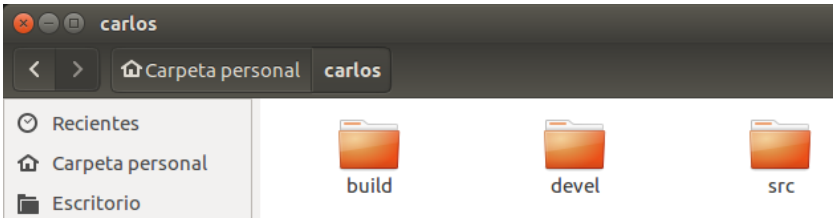

Figura 3 Subcarpetas del Espacio de Trabajo

La carpeta más importante será la llamada “src” que es la abreviación en inglés de la palabra “source” es decir fuente, ya que es aquí donde se van a crear los paquetes de ROS.

2.3.3 Paquetes de ROS

El software de ROS se organiza en paquetes, los cuales pueden contener diferentes nodos, librerías, datos e información para un determinado proyecto. Para ingresar en esta carpeta se utiliza el comando:
$ cd src
Y una vez dentro de esta carpeta se va a crear el paquete del proyecto que se ha llamado “Sistema Cameres”. Esto se realiza con el comando:
$ catkin_create_pkg sistema_cameres std_msgs rospy roscpp

En este comando cabe destacar que además del nombre del paquete se han definido dependencias, es decir prerrequisitos que el paquete buscará para su correcto funcionamiento, con las librerías de mensajes estándar, ROS para python y ROS para c ++ respectivamente.

En la figura 4 se puede observar que una vez ejecutado el comando, se ha creado dentro de la carpeta “src” la carpeta del paquete “sistema_cameres” y un archivo txt llamado CMakeLists, el cual es un archivo protegido que

describe como construir el código y donde instalarlo, además de algunas otras características importantes, como la versión de ROS.

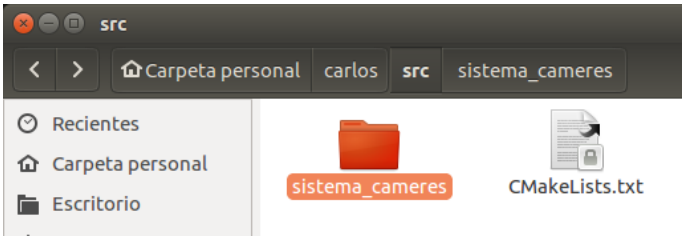

Figura 4 Paquete de Ros creado

Dentro del paquete creado se puede observar, en la figura 5, que se han creado automáticamente 2 subcarpetas llamadas include y src en las cuales se van a almacenar la información que el programa necesita y el código fuente que va a compilar respectivamente. Además de otro archivo CMakelist, el cual tiene la misma función que el archivo CMakeLists antes descrito, y un archivo xml llamado “package” el cual define propiedades como el nombre del paquete, el número de versión, el autor, el encargado de mantenerlo y las dependencias con otros paquetes de catkin.

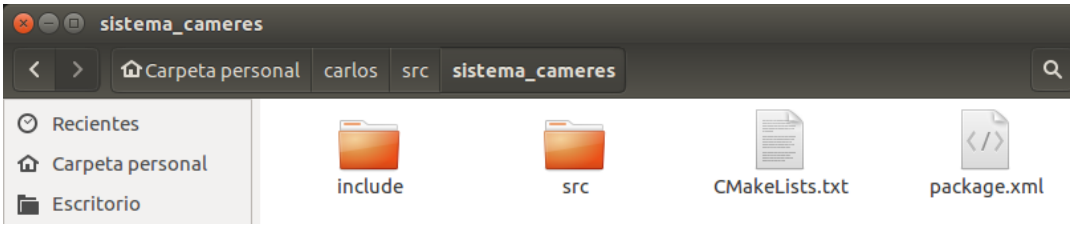

Figura 5 Interior del Paquete

2.3.4 Creación de Nodos Ejecutables

Una vez que se ha definido adecuadamente el paquete de ROS, se pueden ingresar dentro de la carpeta src todos los archivos c++ que se necesiten, como se observa en la figura 6.

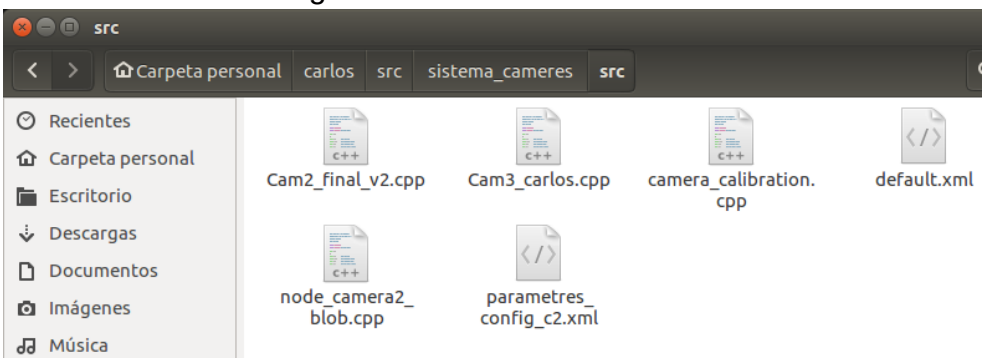

Figura 6 Nodos creados

Sin embargo, antes es importante hablar de 2 temas de la estructura de ROS que no se habían explicado. El primero es el concepto de nodo y tópico. Y el segundo es el concepto de mensaje y sus diferentes tipos.

Un nodo dentro de la terminología de ROS es un proceso que realiza un cómputo, es decir es un programa ejecutable que se comunica con otros programas usando tópicos, servicios y parámetros. Cada nodo posee un nombre único en la red, y en el caso de que se añaden a la red tópicos con el mismo nombre, el maestro cerrará el nodo antiguo y mantendrá en la red el nodo actual. El tópico a su vez es el nombre del bus sobre el cual un nodo intercambia mensajes, a través de una arquitectura publicador/subscriptor, teniendo en cuenta que la comunicación del tópico es unidireccional, es decir que funciona únicamente como subscriptor o como publicador.

En esta arquitectura cada nodo puede estar publicando la información que desea comunicar, sin necesidad de que exista al menos un subscriptor de esta información y presenta también la ventaja de que muchos nodos diferentes pueden estar subscritos al mismo nodo publicador sin que se interfieran unos con otros.

Un mensaje por su parte es una estructura simple de datos, que comprende unos campos específicos, pudiendo ser desde campos simples, o también llamados primitivos, como números enteros, decimales, coma flotante, booleanos y demás, así como arreglos más complejos de distintos campos primitivos anidados en distintos niveles.

Para poder compilar los archivos de programa adecuadamente, y crear nodos ejecutables, se debe editar el archivo de texto “CMakeList” que se encuentra dentro del paquete, junto con el archivo xml “package”. Dentro de este archivo, que se ha generado automáticamente y que se ha llenado parcialmente, se debe ingresar el siguiente código en la parte inferior del archivo de texto:

```
add_executable (camera2 src/Cam2_final.cpp)
target_link_libraries(camera2
${catkin_LIBRARIES}${OpenCV_LIBRARIES})

add_executable(proba_estructura src/node_camera2_blob.cpp)
target_link_libraries(proba_estructura ${catkin_LIBRARIES}
${OpenCV_LIBRARIES})
```

Ya que aquí se especifica que se cree un nodo ejecutable llamado “camera2”, el cual funcionará de acuerdo al código c++ escrito en el archivo “Cam2_final.cpp” y que incluye las librerías de catkin y de OpenCv. La ubicación del código se puede observar en la figura 7.

```
#############
## Testing ##
#############

## Add gtest based cpp test target and link libraries
# catkin_add_gtest(${PROJECT_NAME}-test test/test_sistema_cameres.cpp)
# if(TARGET ${PROJECT_NAME}-test)
#   target_link_libraries(${PROJECT_NAME}-test ${PROJECT_NAME})
# endif()

## Add folders to be run by python nosetests
# catkin_add_nosetests(test)

add_executable(camera2 src/Cam2_final_v2.cpp)
target_link_libraries(camera2 ${catkin_LIBRARIES} ${OpenCV_LIBRARIES})

add_executable(proba_estructura src/node_camera2_blob.cpp)
target_link_libraries(proba_estructura ${catkin_LIBRARIES} ${OpenCV_LIBRARIES})
```

Figura 7 Declaración de Nodos ejecutables

Es importante que en este archivo se especifique que el proyecto tiene dependencia de las librerías de OpenCV, añadiendo en la parte inicial del mismo el siguiente código como se observa en la figura 8.

find_package(OpenCV)
find_package(catkin REQUIRED COMPONENTS
roscpp
rospy
std_msgs
genmsg
geometry_msgs
)

include_directories(
${catkin_INCLUDE_DIRS} ${OpenCV_INCLUDE_DIRS}
)

```
cmake_minimum_required(VERSION 2.8.3)
project(sistema_cameres)

## Add support for C++11, supported in ROS Kinetic and newer
# add_definitions(-std=c++11)

## Find catkin macros and libraries
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages
find_package(OpenCV)
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  std_msgs
)
```

Figura 8 Archivo CMakeLists

Ahora ya se puede compilar todo el paquete dentro del espacio de trabajo, mediante el comando

$catkin_make

2.3.5 Ejecución de Nodos

Una vez que se han realizado todos los pasos anteriores, y los nodos se han compilado correctamente, el paquete tiene toda la información necesaria para funcionar adecuadamente. Para poder probar su funcionamiento se debe ejecutar en una terminal el código:

$roscore

Este comando pone en marcha el servicio de ROS, al iniciar un *Master* de ROS, además de llamar una colección de nodos y programas que son prerrequisitos de cualquier sistema basado en ROS. Por tanto, debe estarse ejecutando todo el tiempo, para que los nodos de ROS se puedan comunicar entre sí. Este comando creará un Maestro de ROS (Master) además de parámetros y servicios de ROS y un nodo llamado “rosout” el cual es un mecanismo de registro de todos los eventos de la plataforma ROS.

Este comando sin embargo se ejecuta únicamente una vez en una terminal dentro de la red, ya que el maestro aquí creado es el que utilizarán todos los nodos en diferentes ordenadores. En todos los demás ordenadores de la red, en vez del comando “roscore” se debe utilizar el comando:

export ROS_MASTER_URI=http://192.168.1.57:11311

El cual le indica a la terminal que el maestro de la red ROS, estará en el ordenador con la dirección IP indicada y se conectará a través del puerto

11311, el cual es el puerto de comunicación Ethernet. Consecuentemente es un requisito primordial que todos los ordenadores que se utilicen sean capaces de comunicarse entre sí a través de la red Ethernet.

Cuando ya se ha establecido el maestro de la red se debe abrir una nueva terminal en la cual se deben configurar las variables ambientales del paquete creado, mediante los comandos:

$ cd carlos
$ source devel/setup.bash

Y una vez hecho esto se puede utilizar el comando rosrun, en el cual se especifica el nombre del paquete y del nodo que se desea ejecutar. Por ejemplo en el proyecto se utiliza el siguiente código para ejecutar el nodo "camera2".

$ rosrun sistema_cameres camera2

2.3.6 Cambio de Variables Ambientales

Finalmente, dado que normalmente se trabaja con muchas terminales diferentes, en diferentes ordenadores, cada una ejecutando un nodo en específico, se puede configurar que todas las terminales carguen las variables ambientales adecuadas mediante los comandos.

$ echo source ~/carlos/devel/setup.bash>> ~/.bashrc
$ echo *export ROS_MASTER_URI=http://192.168.1.57:11311*>> ~/.bashrc
$ source ~/.bashrc

Con lo cual se sobreescribe sobre el "bashrc" el cual es un script que se ejecuta cada vez que una nueva terminal es iniciada. De esta forma en cada nueva terminal abierta se añaden las variables ambientales del archivo de configuración del paquete y se establece que el maestro de la red se encuentra en el ordenador con la IP del maestro.

Estos cambios se pueden verificar con el comando

$ echo $ROS_PACKAGE_PATH

El cual debe dar la siguiente respuesta:

/home/control/carlos/src:/opt/ros/kinetic/share

2.4 Matlab

Matlab es un programa matemático ampliamente utilizado por ingenieros y científicos, el cual posee un entorno de desarrollo integrado, es decir todos

los servicios integrales para facilitar el desarrollo de software, y su propio lenguaje de programación denominado lenguaje M. Es una herramienta muy poderosa la cual es ampliamente utilizada en el desarrollo de sistemas de control ya que permite integrar diferentes técnicas tanto de control moderno como técnicas de control avanzados.

En el presente proyecto se encargará de recibir la información publicada en tópicos ROS para realizar el control del helicóptero. La estructura de control se explicará con más detalle más adelante.

2.4.1 Conexión de Matlab con ROS

Actualmente Matlab cuenta con una interfaz entre Matlab/Simulink, con ROS, la cual permite la comunicación con una red ROS, explorar las capacidades de la red e intercambiar datos en tiempo real. Adicionalmente permite generar redes ROS internas dentro de Matlab y Simulink, así como importar archivos de sesión de ROS llamados rosbags, en los que se almacena información publicada en un tópico, entre otras muchas características. Todo esto se realiza a través de la toolbox “Robotic System Toolbox” la cual ha sido instalada en los ordenadores del laboratorio de sistemas de control avanzado.

Una vez que se ha instalado el *toolbox* necesario, se debe configurar el maestro de la red ROS, el cual matlab permite tanto crear un nuevo maestro, como también conectarse a un maestro ya existente en la red. El comando necesario es

rosinit

Este comando además de iniciar o conectarse a un maestro en la red, crea un nodo global de Matlab, quien está conectado al nodo maestro y se puede emplear en otros programas. En el presente trabajo se va a conectar el ordenador con matlab, al nodo maestro que está en el séptimo ordenador y se ha configurado con la dirección IP ‘192.168.1.57’, de tal forma que el comando completo en el *workspace* de matlab será

rosinit(‘http://192.168.1.57:11311’)

Antes de ejecutar este comando, sin embargo, es recomendable verificar que los ordenadores que se encuentran en una misma red, posean cada uno un archivo host válido, ya que, de no hacerlo, habrá un problema en la comunicación con Matlab de forma que los nodos nunca llegarán a subscribirse unos con otros. Este archivo “hosts” se utiliza para traducir los

nombres de dominio con las respectivas direcciones IP, de forma que la red podrá conocer a que ordenador debe conectarse. Este archivo dentro de Windows se encuentra en la dirección C:\Windows\System32\drivers\etc y debe configurarse de la siguiente manera: Primero debe ir la dirección IP del ordenador para a continuación en la misma línea el nombre del dominio que se desee asignar.

En la figura 9 se observa como se ha configurado este archivo para el laboratorio de control avanzado de manera que se mantengan los antiguos nombres de cada ordenador.

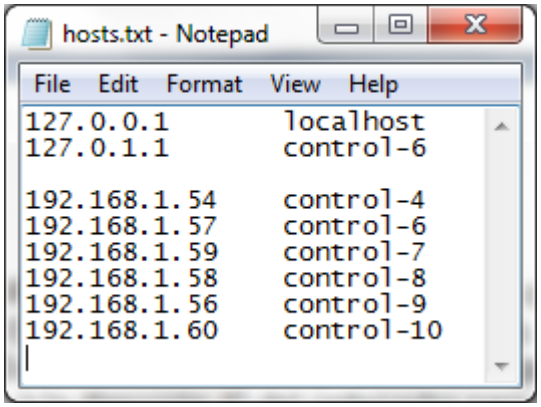

Figura 9 Archivo Hosts Windows

Este proceso debe realizarse en todos los ordenadores con sistema operativo Ubuntu para garantizar una correcta comunicación entre todos los equipos. Este archivo se puede editar directamente escribiendo el siguiente comando en una terminal

$sudo gedit /etc/hosts

Y debe quedar de manera similar al archivo de Windows realizando el cambio correspondiente en el nombre del *local host* de cada ordenador.

2.4.2 Creación de publicadores y subscriptores con matlab

El *toolbox Robotic System* permite crear en el *workspace* de Matlab, tanto nodos publicadores como nodos subscriptores, mediante los comandos *rospublisher* y *rossubscriber* respectivamente. En nuestro caso para realizar pruebas de comunicación de datos desde los nodos de ROS hacia el ordenador con Matlab, se definió un nodo subscriptor el cual se conecta al nodo de la cámara 2.

El código de matlab es el siguiente:

posesub = rossubscriber('/camera2)

Después de eso existe otro comando en MATLAB el cual al ser llamado permite recibir información de un tópico en particular. Este comando se llama receive y se debe especificar el nombre del nodo subscrito, así como también un tiempo límite (timeout) que determina el tiempo máximo de espera a la llegada de información antes de dar por terminado el intento de comunicación. El código de Matlab es el siguiente:
posn=receive(posesub,10);

2.4.3 Creación de publicadores y subscriptores con simulink

Estos comandos funcionan muy bien en scripts, sin embargo, cuando se trabaja con Simulink, existen también como parte del *toolbox* unos bloques especiales para la comunicación con ROS. Estos bloques al igual que los comandos de *script*, permiten crear tópicos publicadores y subscriptores que envíen y reciban mensajes en la red ROS, crear mensajes en blanco de un determinado tipo y también recibir y establecer parámetros de un servidor ROS. En este caso se va a utilizar un nodo subscriptor como se observa en la figura 10.

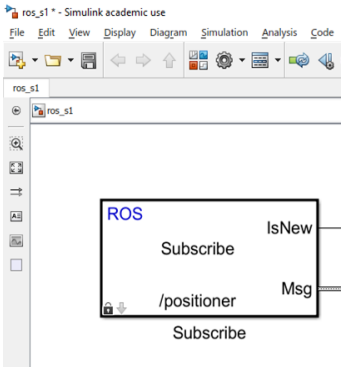

Figura 10 Bloque de Subscribirse a un Tópico

Dentro de este bloque se debe especificar el tópico de la red ROS del cual se desea recibir información. Sólo aparecen disponibles los tópicos que se encuentren en la red en ese momento por lo cual es importante tener una red ROS funcionando con al menos un tópico activo al momento de utilizar este bloque, ya que de otra forma no será capaz de subscribirse a ningún lugar.

Adicionalmente en este bloque hay 2 salidas diferentes. La primera es el valor que ha recibido de la red ROS, mientras que el segundo es un booleano que indica si el dato recibido es un dato nuevo o no.

Entonces se decidió crear un bloque que se subscribe al tópico de la cámara 1, el cual publica la información de la posición y orientación que ha calculado. Este resultado se envía al *workspace* de matlab y después de separarlo y ordenarlo debidamente, se lo puede graficar obteniendo el resultado de la figura 11 en donde se observan los diferentes datos que se han recibido del tópico publicador de la cámara 1 durante 30 segundos.

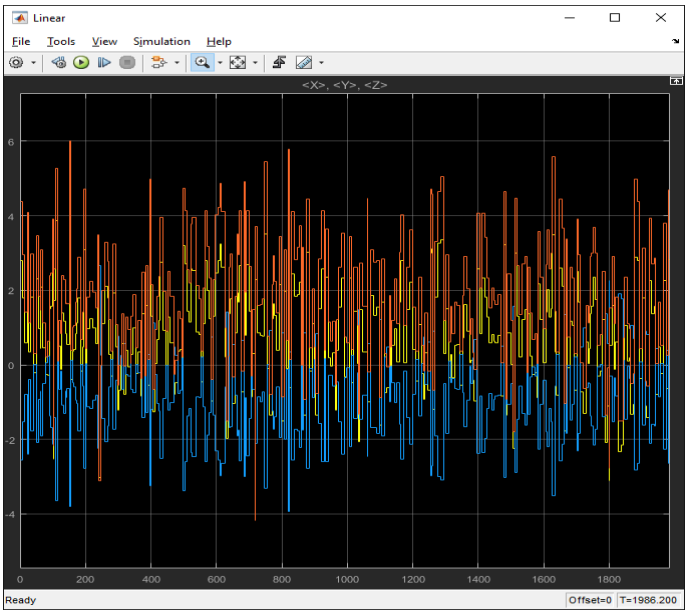

Figura 11 Información Nodo Central

En este gráfico se observa en color amarillo los datos de la posición en el eje x, en color azul los datos de la posición en el eje y, y en color naranja los datos de la posición en el eje z, todos expresados en milímetros.

3 SISTEMA DE VISION

En este capítulo se va a explicar el funcionamiento del sistema de visión artificial, y cuáles fueron los resultados obtenidos, una vez que todos los diferentes programas fueron compilados para funcionar en las versiones actuales de software como se explicó en el capítulo anterior.

Es importante entender el concepto del término “Blob” ya que se va a utilizar mucho a lo largo del capítulo Un blob es un conjunto de pixeles contiguos en una imagen, los cuales comparten alguna característica en común. Esta característica puede ser un valor binario, un valor en escala de grises o en un canal RGB etc. En este caso deben ser pixeles de color claro, tener una distancia mínima de 1 pixel y un área en la imagen comprendido entre 30 y 1500 pixeles cuadrados.

Existen 3 programas que son importantes al momento de trabajar con la visión artificial. El primero que se utiliza en todas las cámaras se llama “Proba_estructura” y una vez que se ejecuta, se encarga de capturar imágenes desde la cámara de video, luego realiza una detección del número de blobs, los cuales deben cumplir ciertas características de tamaño y forma, los muestra en color rojo, en una ventana con la imagen que recibe de la cámara, y finalmente imprime esta información en la pantalla del ordenador. Este programa no crea ni utiliza nodos de ROS, sino únicamente librerías de OpenCV.

El segundo programa que existe en cada cámara se llama “Camera” y el número de la cámara correspondiente (1, 2, 3 o 4) y funciona de la siguiente manera: Una vez que se ha iniciado el nodo, este programa muestra el mensaje en pantalla “Esperando orden de Inicio”. Esto se realizó para poder sincronizar el inicio de la toma de datos de la posición inicial, de forma que cuando se trabajen con más de una cámara, todas ellas puedan iniciar al mismo tiempo y que el cálculo de los desplazamientos, con respecto a esta posición, sean similares entre sí.

Una vez que el nodo de inicio ha enviado la orden, cada cámara comienza a capturar imágenes, identificar blobs en cada imagen, y si el número de blobs es el correcto, es decir 4 blobs, entonces se ordena esta información por el tamaño de los blobs; se calculan vectores de posición y rotación y se almacena este valor para que una vez que se obtengan 120 imágenes, se calcule el promedio de estos valores y sean almacenados como la

información inicial. A esta información inicial se le hace un tratamiento adicional mediante la función Rodriguez para facilitar el cálculo de los ángulos de rotación que se realiza después.

Para realizar el cálculo de la posición se utilizan 4 puntos en el espacio. Primero se crean triángulos entre el centro óptico y 2 puntos del objeto y se expresa la distancia entre ellos como las longitudes de los lados de un triángulo. En donde esta longitud se calcula utilizando la ley del coseno. Este proceso se repite con otras combinaciones para obtener un sistema de ecuaciones, que se resuelve con la condición de que los ángulos deben estar comprendidos entre 0 y 90° y las distancias deben ser positivas.

Este método de cálculo se denomina método de 3 puntos, que es el valor mínimo de puntos para resolver el sistema, y se implementa en OpenCV con la función *solvePnP*. Esta función requiere un vector con las coordenadas físicas del objeto (objpoints), un vector con las coordenadas cámara del objeto (impoints) y los coeficientes de distorsión de la cámara (k) para devolver como resultado un vector de rotación (rvec) y un vector de traslación (tvec).

Este vector de rotación es un vector en formato ejes/ángulo, que es la mínima representación de una rotación 3D, por lo cual necesita ser transformado en forma de matriz, para facilitar el cálculo de los ángulos de rotación. Esto se realiza en OpenCV con la función “Rodriguez” la cual requiere un vector de rotación (rvec) y devuelve una matriz de rotación (rmatrix), empleando la fórmula de Rodriguez para resolver esta transformación. Esta fórmula no es única, sino que se puede expresar la misma rotación de distintas maneras.

Con esta información inicial almacenada en cada cámara, se muestra en pantalla junto con el mensaje “Inicialización completada” y se pasa a la siguiente parte del programa en la cual se realiza el mismo procedimiento anterior, capturando una imagen, encontrando sus blobs, ordenándolos por tamaño, calculando los vectores de posición y rotación, pasando de un vector a una matriz de rotación, y una vez aquí, se calcula la nueva posición y orientación de los marcadores con respecto a la información inicial. De esta forma a pesar de que cada cámara tenga unos vectores iniciales diferentes, entre sí, al momento de calcular el desplazamiento o rotación, es capaz de calcularlo con un error menor al 5%.

Finalmente, el último programa se llama “nodo central” y a diferencia de los otros programas no es independiente para cada cámara, sino que existe uno solo en la red. Este programa se encarga de subscribirse a los tópicos de las cámaras, recibir la información como un mensaje para convertirla de un valor de cadena a un valor decimal, calcular el promedio de los valores de las diferentes cámaras e imprimir estos valores promedio de posición y orientación en la pantalla.

3.1 Pruebas Realizadas

En la primera prueba que se realizó se utilizó solamente una cámara, y se comprobó en primer lugar que las condiciones de luz sean las adecuadas mediante el programa “Proba estructura”. El resultado de este programa se puede observar en la figura 12 donde se observa que las condiciones de luz son adecuadas ya que se identifican claramente 4 blobs en la imagen.

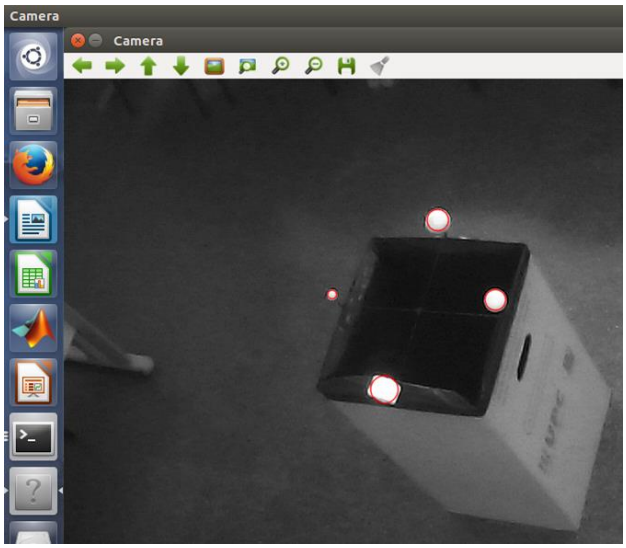

Figura 12 Nodo Proba Estructura

Después de comprobar las condiciones de luz, se ejecuta el programa de la cámara 1 junto con el programa “nodo central” y se observa el resultado en la pantalla. Este resultado se va a imprimiendo constantemente en la pantalla del ordenador con lo cual resulta imposible de analizar su comportamiento. Aquí se deberá realizar un cambio para que la información de la posición y orientación pueda ser leída en Matlab para ahí ser analizada.

3.2 Cambio en el Programa Nodo central

En el programa “nodo central” el nodo creado, solamente se subscribe a los tópicos que publican las cámaras para recibir esta información, procesarla y mostrarla en pantalla, sin embargo, no tiene configurado dentro de su ejecución un tópico publicador para poder trasmitir esta información al resto de la red, como por ejemplo el ordenador con matlab.

Adicionalmente los mensajes que se trasmiten a través de estos tópicos de la cámara, son mensajes de tipo cadena de caracteres, en los cuales se ha ingresado toda la información de la posición y orientación calculada, en el orden: x, y, z, pitch, roll, yaw. Separados entre sí con comillas dobles. Esto aumenta los tiempos de procesamiento, ya que además del tiempo necesario para recibir un mensaje, se debe también leer el mensaje, separarlo en partes y transformarlo de cadena de caracteres a un número decimal.

Por este motivo se decidió en el nodo central, añadir un tópico de tipo publicador, el cual trasmita los valores promedios que ha calculado, y lo haga a través de un mensaje de tipo “geometry_msgs/Twist”.

Este tipo de mensaje es una estructura anidada en la que se han creado 2 vectores, el primero llamado “linear” y el segundo llamado “angular”, que se utilizan para expresar la velocidad, tanto en sus componentes lineales x,y,z, como en sus componentes angulares que también se los representa con x,y,z. Sin embargo en el presente proyecto, se les asigna un nuevo significado interno a esta estructura, ya que en el vector lineal x,y,z se enviará la información de las componentes x,y,z de las coordenadas de posición, mientras que en el vector angular x,y,z se enviará la información de los ángulos de rotación, pitch, roll y yaw.

Una vez se han hecho estos cambios en el nodo central, se puede cambiar el diseño dentro de simulink, para que el bloque correspondiente se subscriba directamente al nodo central, y separar la información del mensaje que llega en la parte de la posición y en la orientación. Este cambio en el diseño se observa en la figura 13 donde se observa como el bus se ha separado en 6 componentes de las cuales 3 corresponden a la posición y 3 corresponden a la orientación.

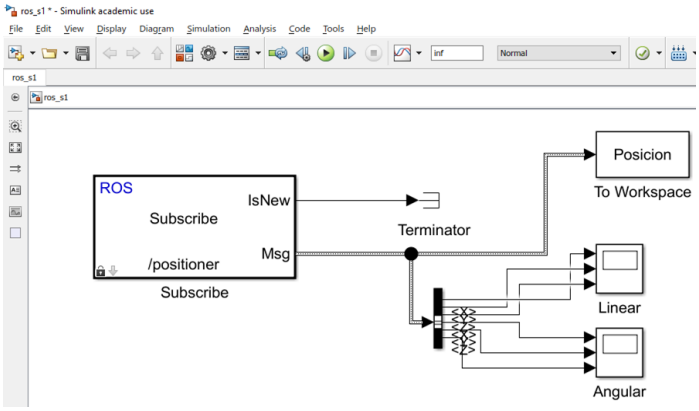

Figura 13 Bloque Conexión directa

Con este cambio se puede observar en la figura 14 los resultados de la prueba con una cámara en los que se movían los marcadores a lo largo de los 3 ejes espaciales y también como se rotaba la estructura de los marcadores para apreciar los cambios de valores de los ángulos pitch roll y yaw.

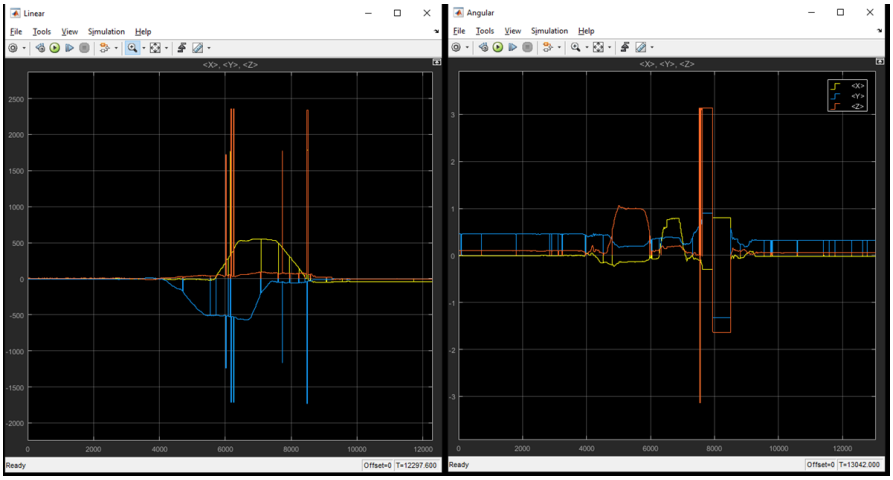

Figura 14 Resultado Pruebas 1 Cámara

Una vez que se obtuvo un resultado adecuado al trabajar con una sola cámara, se procedió a realizar pruebas activando las 2 cámaras que se encontraban instaladas. Adicionalmente se cambió de nuevo el diseño de simulink para que ahora sea un solo Scope, en el que se pueda observar las 6 variables que se obtienen en la visión artificial. El resultado de esta prueba se observa en la figura 15.

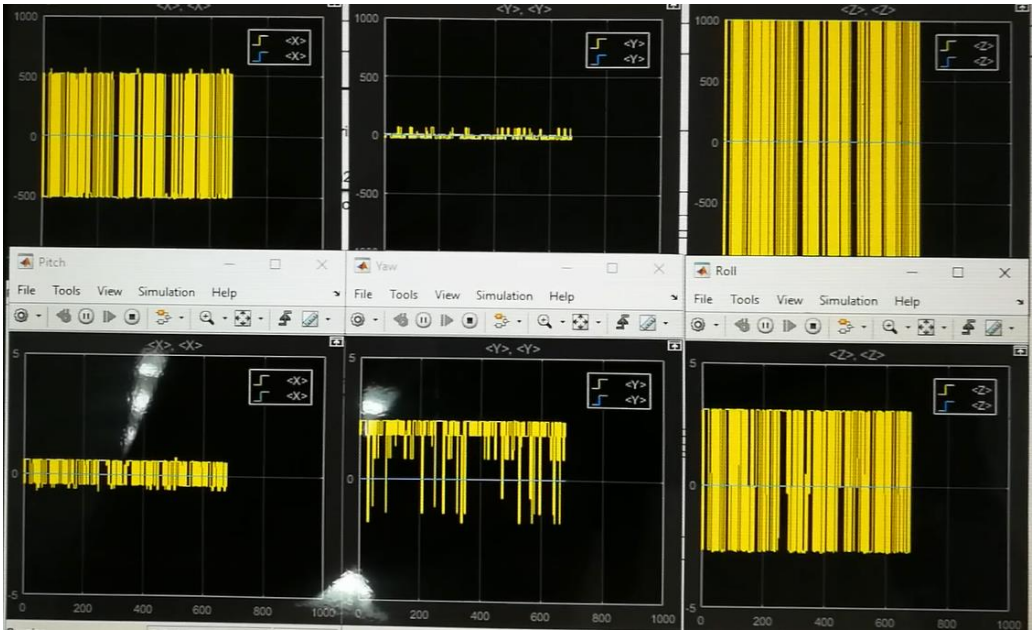

Figura 15 Resultado Prueba 2 Cámaras

Este resultado como se puede observar presenta mucho ruido y oscilaciones por lo que para continuar con el trabajo es necesario analizar los motivos de lo que ha sucedido.

3.3 Análisis de las pruebas

Realizando más pruebas del sistema se ha determinado, que estas discontinuidades de la figura anterior no se producen debido al cálculo del promedio de los valores publicados por 2 cámaras distintas, sino que se producen dentro del programa de la cámara, y dependiendo de ciertas condiciones de iluminación, o de la posición de inicio, pueden afectar a una u otra.

El primer problema que se ha encontrado en el programa de una cámara es un error debido al número de blobs en la imagen, el cual se debe a variaciones en la cantidad de luz del laboratorio, que ocasiona que ciertas zonas de objetos se iluminen en mayor o menor medida, y sean detectados como blobs de la estructura. O a su vez, que no se detecten todos los blobs de la estructura de marcadores. Este problema se podría corregir, quitando toda la luz natural del sistema, ya que varía a lo largo del día, y añadiendo unas fuentes de luz artificial que nos garanticen un nivel de luminosidad constante, sin embargo, la relación costo beneficio de esta solución no es buena ya que implica añadir muchos elementos al sistema.

Una mejor solución tanto en términos prácticos como económicos y que reduce ampliamente el problema, es verificar siempre las condiciones de luz antes de realizar un experimento, y corregirlas, aumentando o quitando luz del sistema y bloqueando posibles reflejos o zonas de luz indeseadas, así como también garantizar el no utilizar esta información en el algoritmo de cálculo de la posición y orientación.

Otro problema que se encontró en la ejecución del programa ocurre cuando el nodo central realiza el cálculo del promedio de las cámaras activas. Se utiliza el valor promedio con el objetivo de reducir el efecto del ruido que puede aparecer en los cálculos de una cámara, lo cual sí se consigue en el caso de pequeñas variaciones del valor ya que estos pequeños picos o valles de una cámara al ser promediados con un valor más estable, se reduce la amplitud de su error. Un ejemplo de esta situación se observa en la figura 16 en donde el valor promedio de las 4 cámaras presenta un error menor que el de las cámaras.

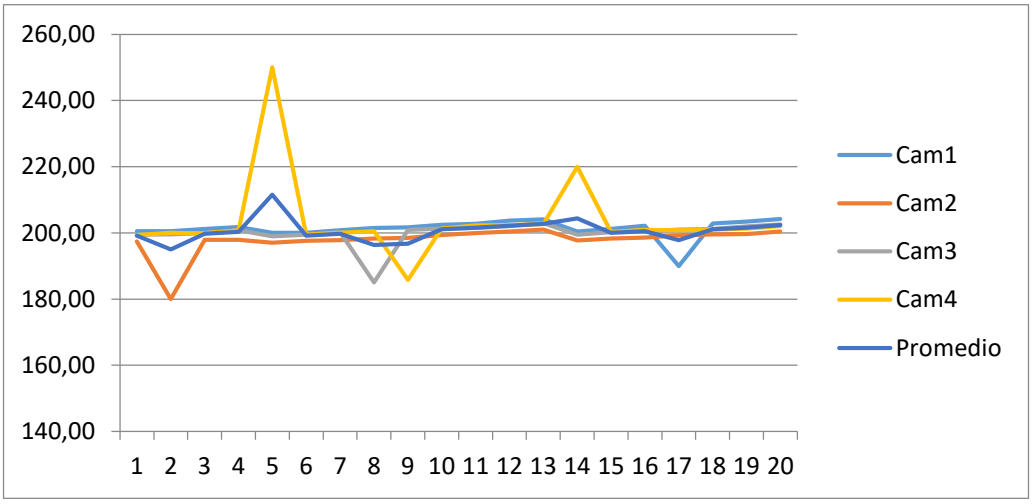

Figura 16 Promedio 4 Cámaras Adecuado

Sin embargo, cuando una cámara genera grandes discontinuidades producto de un error de cálculo, la reducción que se logra del error, no es comparable con lo que se ha afectado al valor promedio. Esto se observa gráficamente en la figura 17 en donde el error de una cámara ha hecho que el valor promedio también posea una dispersión muy alta. De esta forma el promedio en vez de reducir el error de una cámara, ayuda a propagar los errores de cada cámara en el valor del nodo central.

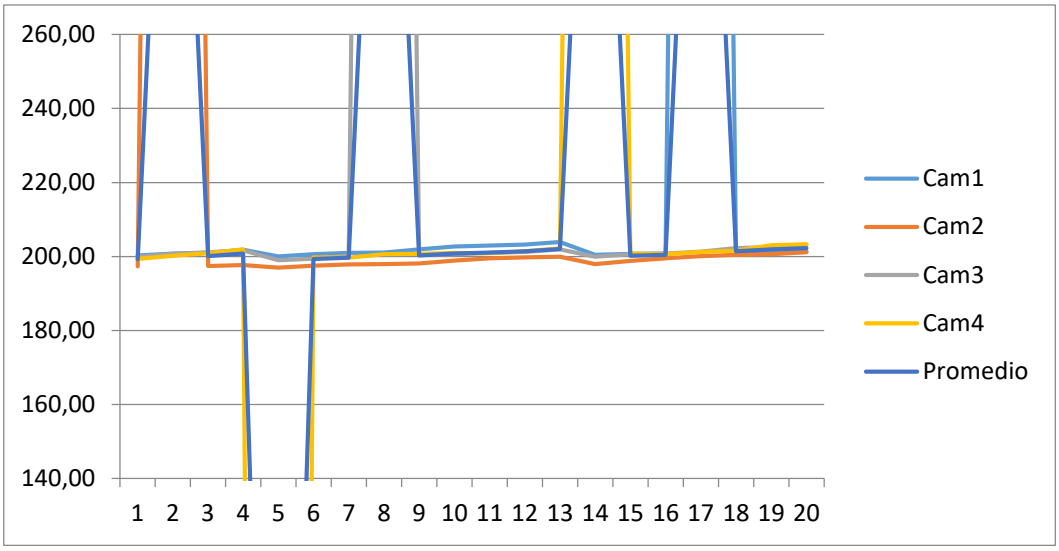

Figura 17 Promedio 4 Cámaras Equivocado

Por este motivo no se puede aplicar directamente la función promedio, sino que se debería utilizar otra función estadística, como la mediana, o a su vez condicionar el sistema para que pueda discernir qué valores considerar para el promedio y cuáles no.

Un tercer problema que se encontró después de realizar diversas pruebas, ocurre en el proceso de inicialización. Como se explicó anteriormente, el programa al momento de encontrar los vectores de rotación y traslación iniciales, va capturando imágenes, detectando blobs y calculando vectores de traslación y rotación, que los acumula para después calcular su promedio. Aquí ocurre un error similar a la situación anterior, donde para pequeñas variaciones, el valor promedio funciona adecuadamente, pero si se introducen vectores con valores muy diferentes del resto de vectores correctos, entonces el promedio no se corresponde a la realidad, sino que se desvía en gran magnitud de este valor.

La solución por tanto es similar al caso del nodo central, en donde se puede probar funciones estadísticas diferentes, o condicionar los valores que se utilizan para calcular el promedio.

Luego se encontró un problema muy significativo en el programa de la cámara, el cual es un error de identificación. Para entender este error hay que explicar cómo funciona el algoritmo de ordenación que lleva el programa. Una vez que se han detectado los blobs en una imagen se tiene la siguiente información dentro del vector “keypoints”:

- Número de Blobs detectados:
- Un número de identificación para el blob detectado
- Tamaño de cada Blob
- Posición en Coordenadas Cámara

La posición en coordenadas cámara se refiere al número de pixeles de distancia, a lo largo de los ejes x,y, que tiene un punto con respecto al eje de coordenadas de una imagen, el cual se encuentra en su esquina superior izquierda. De esta forma a medida que el punto se encuentra más abajo y a la derecha de la imagen, tiene valores x y cada vez mayores. En la figura 18 se puede observar en donde está ubicado el origen de estas coordenadas cámaras y las coordenadas en pixeles de un punto en particular.

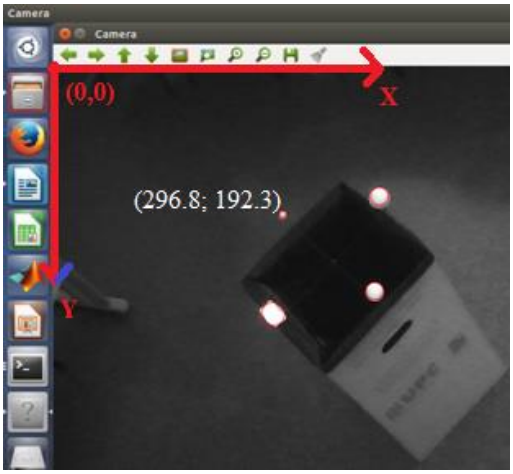

Figura 18 Coordenadas Cámara

Esta información del vector *keypoint* se genera automáticamente y el orden en que se genera puede ser distinto en una misma imagen, por lo cual debe existir un vector en donde las coordenadas de los blobs se encuentren siempre debidamente ordenadas y así se correspondan tanto con los marcadores que representan, como con las coordenadas físicas de estos marcadores.

Esta acción de orden se condiciona para que se realice solamente cuando se han detectado 4 blobs, y de estos 4 blobs la información de sus posiciones en coordenadas cámara se organiza primero el nodo de mayor tamaño, que corresponde con el marcador de forma cúbica, después el de menor tamaño que corresponde con el marcador esférico más pequeño,

tercero el tercer nodo en tamaño que corresponde con el marcador esférico mediano y como último el nodo restante que corresponde con el marcador esférico más grande.

Este algoritmo en ocasiones da buenos resultados, sin embargo, debido a efectos de niveles de luz del ambiente, muchas veces el nodo de mayor tamaño no se encuentra en el marcador de forma cúbica, sino en el marcador esférico más grande. O debido a efectos ópticos, encuentra de igual tamaño a los marcadores esféricos mediano y grande. Estos no son los únicos errores que se pueden generar, sino que pueden existir muchos errores al identificar que coordenadas cámara corresponden a que marcador. Algunas de estas son imposibilidades físicas, o que representan deformaciones o cambios del sentido de los ejes, y ocasionan graves errores al momento de calcular los vectores de traslación y rotación en cada cámara.

Un ejemplo de estos errores de identificación se observa en la figura 19 donde se imprime en pantalla la información interna del programa.

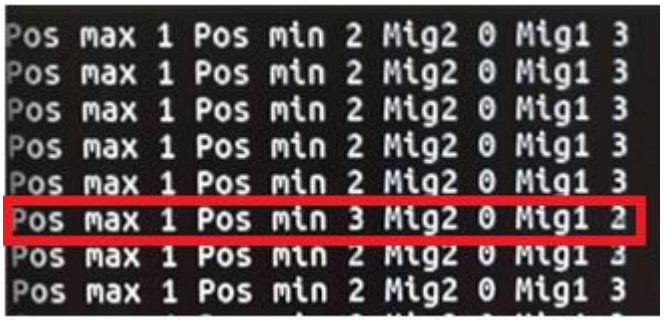

Figura 19 Error de Identficación

En esta figura se está publicando el mensaje *Pos max* junto al número de identificación del marcador de mayor tamaño; *Pos min* junto al número del marcador de menor tamaño; *Mig 2* junto al número del segundo marcador de menor tamaño y *Mig1* junto al número del segundo marcador de mayor tamaño. Y se observa que en la línea resaltada en rojo ha ocurrido un cambio en el orden de estos números, es decir que a pesar de que la estructura física no se mueve, el algoritmo identifica los marcadores de una manera diferente.

Este problema no puede resolverse físicamente, sino que se debe diseñar e implementar un algoritmo en el programa para poder ordenar la

información de una manera adecuada, ya que este error afecta gravemente la robustez del sistema.

Adicionalmente, como se explicó anteriormente, los nodos cámaras envían la información que calculan internamente, a través de un mensaje de tipo cadena de caracteres. Esto hace que los datos de las cámaras no sean directamente accesibles, sino que primero deban tener un tratamiento para ser leídos, separados y finalmente reconvertidos a un valor decimal. Si bien no es un problema que afecte al cálculo del nodo, sí que le añade tareas computacionales, y no permite trabajar cada cámara de manera independiente. Por estos motivos se ha decidido también realizar un cambio de esta arquitectura. La forma de afrontar todas estas situaciones se va a explicar a lo largo del capítulo 4.

4 ROBUSTIZACIÓN DEL SISTEMA DE VISIÓN

En el capítulo anterior se realizaron diversas pruebas del estado del sistema de visión artificial, sin embargo, se han encontrado diversas condiciones que se deben mantener para su adecuado funcionamiento. Es por eso que en este capítulo se van a buscar posibles mejoras que se pueden aplicar para que este sistema sea más robusto, es decir que sea capaz de trabajar en condiciones menos favorables. Se van a explicar los diferentes cambios tanto a nivel físico como del programa en las cámaras.

4.1 Cambios Físicos

El primer cambio que se va a realizar es añadir más cámaras al sistema, ya que en ese momento solo se contaban con 2 cámaras ubicadas de manera diagonal en el techo del laboratorio de control avanzado. Al añadir más cámaras se busca que en los casos donde se pierda la información de alguna de ellas, o sea incorrecta, se puede mediante la información de otras cámaras, encontrar los valores de posición y orientación adecuados, y de ser posible, permitir recuperar la información de la cámara con defectos. Se emplean cámaras infrarrojas ELP-USBFHD05MT-RL36-S con una resolución de 1920 x 1080 pixeles las cuales trabajan a 60fps (frames per second), es decir son capaces de obtener 60 imágenes en un segundo.

4.1.1 Calibración

Antes de instalarlas y poderlas conectar es necesario realizar una calibración de estas cámaras ya que nunca se habían utilizado. La calibración de una cámara requiere calibrar parámetros intrínsecos y extrínsecos. La calibración intrínseca consiste en obtener una serie de parámetros internos de la cámara, como son la distancia focal, la altura y anchura de los pixeles, entre otros, mientras que la calibración extrínseca consiste en una matriz de rotación y un vector de traslación que relacionan las coordenadas de la cámara con el sistema de coordenadas del mundo.

Se va a realizar una calibración intrínseca de las cámaras, que consiste en:

- Determinar la matriz de distorsión de la cámara
- Determinar la matriz de la cámara
- Calcular el error de re proyección
- Almacenar los resultados en un archivo XML

Para realizarlo se puede emplear un programa de ejemplo llamado “camera_calibration.cpp” que se encuentra en las librerías de OpenCV en la dirección:

samples/cpp/tutorial_code/calib3d/camera_calibration/.

Adicionalmente a este archivo se necesita imprimir la figura con la que se va a realizar la calibración, la cual puede ser un tablero de ajedrez o una serie figuras redondas asimétricas. Se escogió utilizar el tablero de ajedrez que se observa en la figura 20.

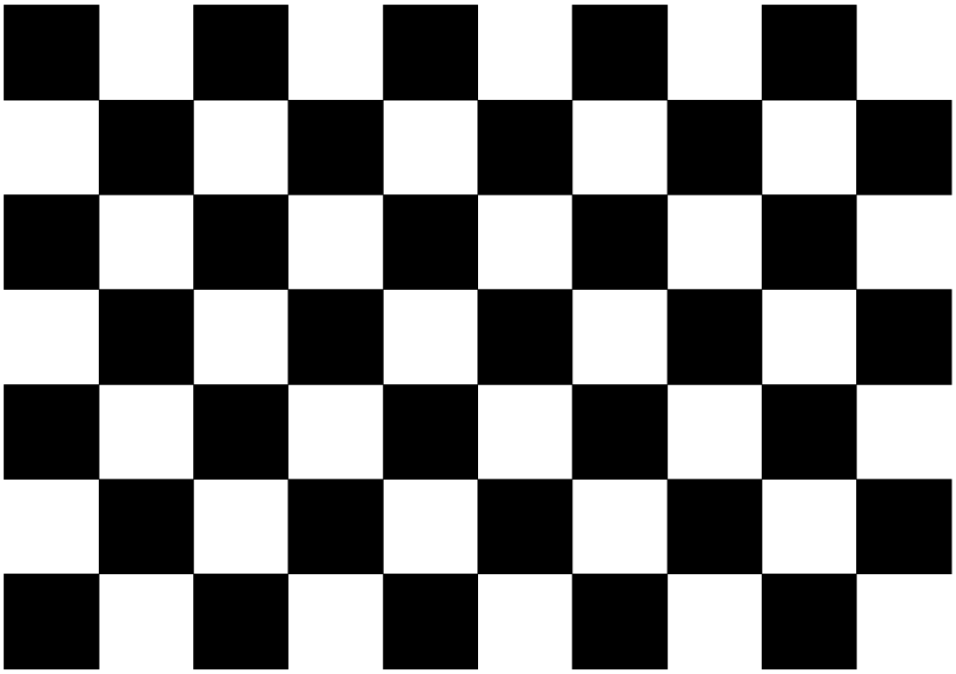

Figura 20 Tablero de Ajedrez

También es necesario que exista un archivo de configuración inicial en el cual se especifican las condiciones físicas de la imagen escogida como son:

- El número de esquina internas a lo largo y a lo ancho de la figura 20, que son 6 y 9 respectivamente.
- El tamaño de los cuadrados en milímetros, en este caso 26 mm.
- El tipo de imagen escogida, “Chessboard”
- Si se utiliza una entrada de cámara, entrada de video o un listado de imágenes, en este caso entrada de cámara.

- El tiempo de espera entre imágenes de la cámara, que se estableció en 100mseg.
- El número de imágenes para la calibración, 25 imágenes.

Con todos estos requisitos se debe, en una terminal nueva, acceder a la carpeta en donde se almaceno los archivos y utiliza el siguiente comando para compilar el archivo de calibración .cpp

$ g++ camera_calibration.cpp –o Calibrar

Una vez que se ha compilado el programa se observa que dentro de la carpeta de trabajo se ha creado un archivo ejecutable que al ser invocado nos despliega una ventana en donde se observan las imágenes que está capturando la cámara como se observa en la figura 21 junto a la pantalla del ordenador.

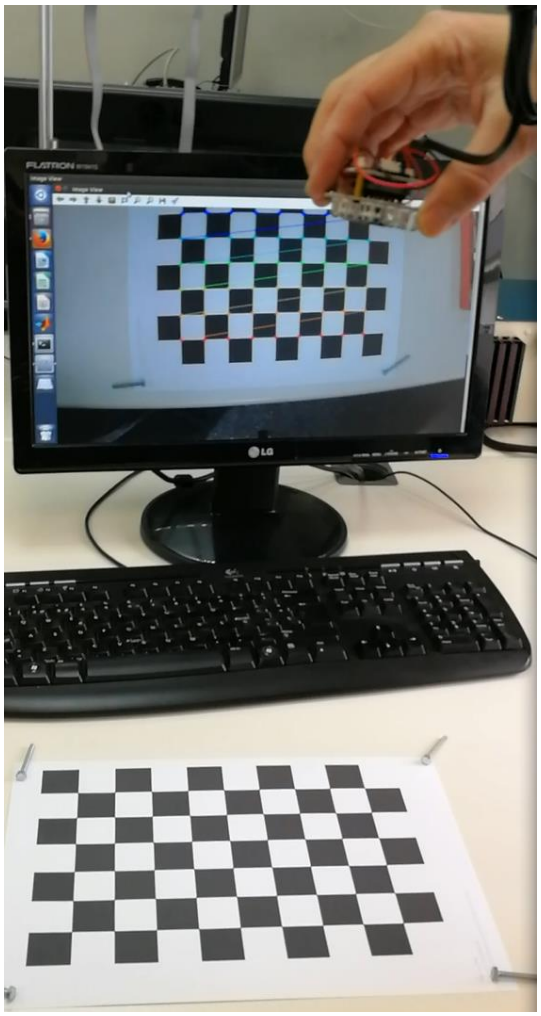

Figura 21 Calibración Intrínseca

El proceso de calibración consiste en mover la cámara sobre la imagen impresa del tablero de ajedrez, en donde irá detectando las esquinas del tablero de ajedrez. De esta forma el programa puede calcular distancias en pixeles en la imagen, para compararlas con distancias conocidas del tablero y determinar así los parámetros antes mencionados.

4.1.2 Instalación

Una vez que se han calibrado adecuadamente las cámaras, se las va a instalar en el techo del laboratorio de control avanzado, y se las debe cablear hasta un ordenador independiente para cada cámara como se observa en la figura 22.

Figura 22 Instalación de las cámaras

Sin embargo, debido a la gran distancia que existe entre la ubicación de las cámaras y los ordenadores del laboratorio, es necesario utilizar cables tipo usb de larga distancia, los cuales tienen una longitud máxima de hasta 10 m y cuentan con un sistema de retroalimentación de la corriente, para garantizar su adecuado funcionamiento.

Estos cables sin embargo no pueden estar conectados al ordenador al momento de su arranque, ya que, debido a esta realimentación de la corriente, el ordenador lo interpreta como un pendrive externo, e intenta arrancar el sistema operativo desde esta conexión usb, y al no encontrar

información del arranque, no es capaz de iniciar ningún sistema operativo ni de realizar ninguna acción.

En la figura 23 se observa la ubicación de las diferentes cámaras en una vista superior del laboratorio de control avanzado, cual es el nombre que se ha dado a cada una de las cámaras comenzando en la esquina superior derecha y se numera en sentido horario. También se observa en un color diferente a cual ordenador se ha conectado cada cámara.

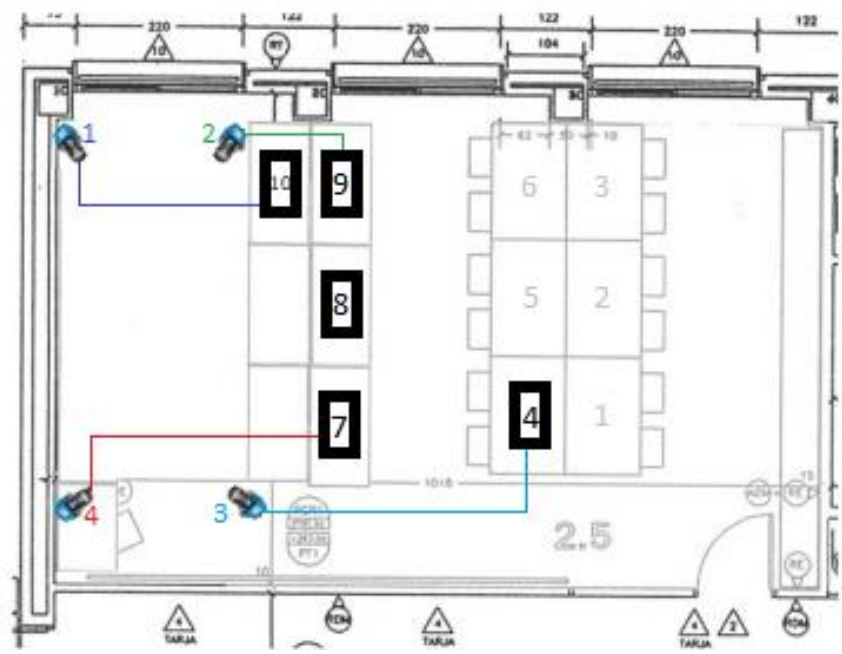

Figura 23 Distribución física de las cámaras

4.1.3 Cambios de los marcadores

Finalmente se va a realizar un cambio en uno de los marcadores de la estructura, para solucionar el problema que existe con el tamaño en píxeles del marcador con forma de cúbica. Este problema se debe a que el tamaño de este marcador puede variar considerablemente, ya que cuando esta de frente a la cámara, se reconoce toda la cara plana del cubo y se obtiene un valor muy alto, sin embargo, cuando el marcador se ubica diagonalmente frente a la cámara, solo se reconoce una parte de la esquina como un blob y su tamaño disminuye hasta ser menor que el marcador redondo de mayor tamaño.

Por este motivo se va a sustituir este marcador por uno de tipo esférico, con un tamaño mucho mayor, de tal forma que independientemente de su

ubicación, el tamaño del blob detectado sea siempre similar. El resultado de este cambio se observa en la figura 24

Figura 24 Marcadores Esféricos

4.2 Cambios en el programa

Una vez que se han implementado los cambios físicos antes descritos es necesario realizar cambios en el programa, para que se pueda mejorar la conexión de todas las cámaras en el sistema e implementar algoritmos de validación de los datos obtenidos. A continuación, se van a detallar los diferentes cambios implementados.

4.2.1 Inicialización

En el capítulo 3 se detalló el problema de la inicialización en el que los valores erróneos en los vectores de rotación y traslación iniciales, introducen un gran error en el valor inicial promedio. La primera opción para resolver este problema es cambiar el método de cálculo estadístico, cambiando el promedio que se ve afectado por errores numéricos grandes, por otro método como la mediana.

Esta solución es capaz de corregir el valor inicial cuando los valores de estos vectores son similares entre sí la mayor parte del tiempo, y en pocas ocasiones se ven afectadas por grandes valores erróneos. Sin embargo, esta condición no se cumple en todas las ocasiones, ya que, debido a pequeños cambios en el orden de los marcadores, se generan grandes errores en el cálculo de los vectores de traslación y rotación, muy diferentes entre sí.

Esto ocasiona que la mediana pueda no estar siempre en el valor inicial correcto, sino más cerca de un valor erróneo que se repite con mucha frecuencia en el proceso de inicialización. Por tanto, un requisito que se

debe cumplir es que todos los vectores de traslación y rotación iniciales sean similares entre sí.

Entonces se va a utilizar un algoritmo para validar que los vectores sean similares entre sí. Esto se realiza almacenando los valores anteriores y comparándolos con los valores siguientes. Si la diferencia entre estos vectores es lo suficientemente pequeña, estos vectores se pueden considerar válidos y por tanto utilizar en el cálculo, mientras que si esta diferencia es alta no se debe tomar en cuenta el valor nuevo, ya que es el que no se ha validado antes, y se descarta para comparar con el siguiente.

Esta solución da un muy buen resultado al momento de determinar los vectores iniciales siempre y cuando la primera imagen que se tomó como referencia haya sido correcta. Esto no se puede garantizar mediante algún algoritmo, y en los casos en que esta primera imagen es errónea ocasiona que no se valide ninguna imagen posterior, ya que si las demás no son incorrectas ninguna se parecerá a la primera, y así nunca se llega a cumplir el número de imágenes válidas necesarias para terminar el proceso de inicialización.

Es por este motivo que se ha limitado el número de intentos de inicialización a 300 imágenes, después de lo cual el programa indica por pantalla el mensaje “No se pudo iniciar” y cierra el nodo creado mediante el comando Rosshutdown.

Finalmente, en los casos en que sí se cumplen las condiciones para inicializar la cámara se muestran en pantalla los valores iniciales de los vectores de traslación y rotación, y adicionalmente el número de imágenes buenas y el total de imágenes utilizadas para que el usuario pueda decidir o no sí es una inicialización válida y continuar con el experimento.

4.2.2 Algoritmo de Orden

El segundo cambio a nivel de programa que se debe realizar, consiste en un cambio en el algoritmo de que ordena los marcadores en el programa. Como se explicó en el capítulo 3, originalmente el algoritmo consistía en que una vez que se han detectado exactamente 4 blobs, estos sean ordenados de acuerdo a su tamaño en este orden:

1. El blob de mayor tamaño
2. El blob de menor tamaño
3. El segundo blob de menor tamaño

4. El segundo blob de mayor tamaño

Este orden corresponde al orden en que están almacenadas las coordenadas físicas de los marcadores, que se observa en la tabla 3, y se establece de acuerdo al sistema de coordenadas propia de los marcadores, que se puede observar en la figura 25.

Tabla 3 Coordenas Mundo en mm

Coordenada	X	Y	Z
Cubo	0	205	35
Esfera Menor	205	0	52
Esfera Mediana	-210	0	40
Esfera Mayor	0	-210	55

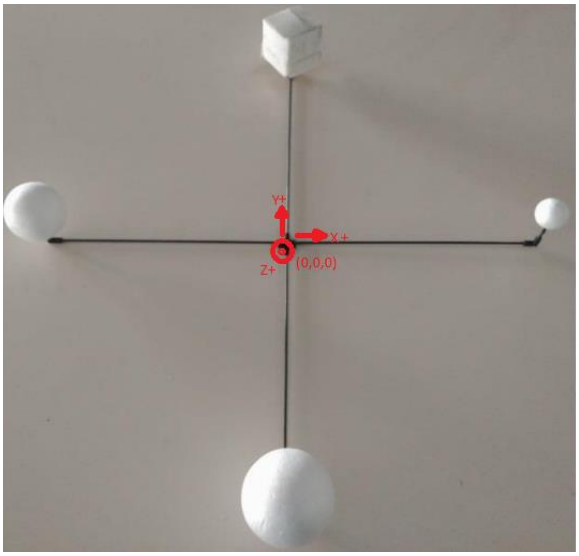

Figura 25 Sistema de Coordenadas del Mundo

Cuando los tamaños de los blobs detectados se corresponden exactamente con el orden de los marcadores, el algoritmo funciona adecuadamente y todos los cálculos de posición y orientación son correctos. Sin embargo, esto no se cumple en la mayoría de situaciones, ya que estos marcadores van a estar sobre el aire, moviéndose y rotando sobre diferentes ejes, por lo cual, en muchos casos, debido a la óptica, o a la cantidad de luz del ambiente, ocurren muchas identificaciones erróneas, como por ejemplo cuando el blob de mayor tamaño no corresponde con el marcador de forma cúbica sino al marcador esférico más grande.

Por este motivo no se puede utilizar este tipo de algoritmo para realizar pruebas de vuelo, sino que se debe utilizar un método diferente. El método planteado consiste en utilizar el algoritmo de orden anterior, una sola vez, en vez de estar utilizándolo constantemente con cada imagen, y con esta primera información ubicar cuales son los marcadores más cercanos en cada momento, para ordenarlos de acuerdo al más próximo.

Esta idea se puede observar en la figura 26 en donde en color rojo se observan los blobs identificados en una imagen. Y encerrados en color verde se observan los blobs de una imagen siguiente.

Figura 26 Sistema de Coordenadas del Mundo

Aquí se observa que, si escogemos el blob de mayor tamaño de los blobs en rojo, éste se corresponde con el marcador de forma cúbica, mientras que, si se escoge el blob de mayor tamaño de los blobs en color verde, éste se corresponde con el marcador esférico más grande. Esto ocasiona que, entre la primera y la segunda imagen, la posición del blob de mayor tamaño se haya desplazado mucho y por tanto en el cálculo de la posición en coordenadas mundo este desplazamiento será mucho mayor. Este salto brusco en la posición del sistema se observa en las gráficas como si se tratase de una discontinuidad en el cálculo.

Por otro lado, si analizamos los blobs en verde que se encuentran más cercanos a los blobs en rojo, podemos ver que se corresponden con el mismo marcador, sin importar el tamaño en pixeles del blob que se haya detectado, por lo cual es una solución adecuada, siempre y cuando se actualice constantemente la imagen de referencia.

El problema ahora consiste en encontrar una manera matemática de determinar cuáles son los puntos más cercanos en cada imagen. Para esto se va a partir de la información disponible en el vector keypoints, específicamente las coordenadas cámara de los blobs detectados, la cual se genera automáticamente al detectar los blobs en una nueva imagen. Por otro lado, utilizaremos la información de las últimas coordenadas cámara que se validaron. Esta validación se realiza de 2 formas. La primera vez se utiliza el algoritmo de ordenamiento por tamaño ya que no existe información previa con la que contrastar y todas las veces posteriores se validan una vez que se ha ordenado adecuadamente los blobs más cercanos con los blobs de la imagen anterior.

Una vez que se tiene esta información es necesario organizarla de alguna manera que permita identificar adecuadamente cuales puntos se corresponden. Para esto vamos a crear una matriz en la que se va a calcular la distancia entre cada punto. Esto distancia se va a calcular como la distancia entre 2 puntos en el plano es decir con la fórmula:

$$d_{ab} = \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2}$$

Y se almacena en la tabla 4 en donde vemos que las filas representan los blobs válidos, mientras que las columnas representan los blobs nuevos.

Tabla 4 Matriz de Distancias entre Blobs

Nuevo Blob			1		2		3		4	
Blob Válido			x	y	x	y	x	y	x	y
1	x	y	d11		d12		d13		d14	
2	x	y	d21		d22		d23		d24	
3	x	y	d31		d32		d33		d34	
4	x	y	d41		d42		d43		d44	

De esa forma el elemento d21 representa la distancia entre el blob válido 2, con el nuevo blob 1 y el elemento d43 representa la distancia entre el blob válido 4 con el nuevo blob 3 y así respectivamente.

Una vez que hemos calculado todas las distancias entre los diferentes blobs, podemos encontrar la distancia mínima en cada columna, es decir cual blob nuevo se corresponde con cual blob previamente validado y así reordenar el vector, para que se corresponda con el marcador adecuado. Esto se realiza con el comando pushback. El cual construye un vector,

añadiendo información al final del mismo, como se observa en el código a continuación.

$impoints.push_back(Point2f(keypoints[pmin0].pt.x,keypoints[pmin0].pt.y))

De esta forma se garantiza que en el vector “impoints” exista siempre la información de las coordenadas cámara, ordenadas de forma que siempre se refieren al mismo marcador físico.

4.2.3 Evaluación Fiabilidad

El proceso explicado anteriormente se realiza únicamente cuando se detectan exactamente 4 nuevos blobs. Ya que, si se realiza con un número de blobs diferentes, estas distancias no tendrían ningún sentido real. Es por este motivo que también se debe condicionar que la distancia entre blobs sea pequeña para evitar así que después de una oclusión, o reflejo, en el que el programa no trabajará, pero los marcadores se continuaran moviendo, se relacionen blobs que no pertenecen al mismo marcador, solamente por estar más cerca de la última posición válida. Este límite se ha establecido en un radio de 20 pixeles, el cual se encontró experimentalmente que daba buenos resultados.

Estas condiciones nos permiten crear unos criterios de fiabilidad del sistema. El primer y más sencillo criterio viene justamente del número de blobs, el cual debe ser exactamente igual a 4. Si estas condiciones no se cumplen, se puede tratar tanto de un reflejo, como de un ocultamiento de un blob, o incluso pérdida de la suficiente cantidad de luz. En todos estos casos la información resultante no es válida para realizar cálculos y no debe ser utilizada.

El segundo criterio que aparece es el de la distancia entre los blobs válidos y los blobs nuevos. Si se ha sobrepasado la distancia máxima, puede ser el caso que explicábamos anteriormente en que muchos blobs nuevos no fueron aceptados y los marcadores continuaron desplazándose hasta otra posición en la que ya no tiene sentido relacionar las distancias con el blob que corresponde. En ese caso tampoco se debe utilizar esta información para realizar cálculos.

Es entonces necesario que esta información sea accesible tanto al usuario, para que pueda corregir algún aspecto del ambiente, como al programa para que no utilice esta información en los diferentes cálculos. Es por esta razón que se decidió cambiar nuevamente el tipo de mensaje del tópico, a

una estructura más compleja. Esta estructura son los mensajes “TwistStamped” los cuales están compuestos de 2 mensajes: el primero un mensaje tipo “Header” y el segundo de tipo “Twist”. El mensaje tipo twits ya se explicó en el capítulo 3 y se va a utilizar de la misma manera, manteniendo los significados de sus componentes linear y angular. El mensaje tipo Header en cambio, se compone de 3 elementos llamados:

- uint32 seq
- time stamp
- string frame_id

El elemento uint32 seq, es un número entero que se va incrementando secuencialmente, y permite identificar cada mensaje generado con un número diferente. El elemento time stamp, permite incluir la hora tomada del ordenador, en el mensaje, pero no se va a utilizar en este caso. Finalmente, el elemento string_frame_id, permite añadir una cadena de caracteres en el mensaje con las que identificar este mensaje de otros. En este caso se va a utilizar este elemento para indicar la información de la fiabilidad de los blobs actuales, permitiendo publicar los mensajes de “Error_#Blobs” para el primer criterio, el mensaje “Error_Perdido” para el segundo criterio, y “Ok” cuando los datos han sido validados.

Finalmente como la información de la fiabilidad ya se encuentra disponible en cada mensaje publicado, es necesario que el programa también sea capaz de entender esta información y así saber qué información utilizar. Por este motivo se ha añadido una función en el nodo Central, la cual se encarga de leer los mensajes que se reciben de los diferentes nodos de las cámaras y utilizar solamente los mensajes que se encuentran en estado OK.

4.2.4 Pérdida de Blobs

Si bien existe información que no es fiable y por tanto no se puede utilizar para el cálculo de la posición y orientación en un determinado momento, se puede intentar recuperar esta información y por esto se plantearon los siguientes algoritmos. El primero es mantener el nodo funcionando, en vez de cerrarlo con el comando roshutdown, ya que así cuando se solucionen los problemas de luminosidad o reflejos en el ambiente, el nodo puede volver a trasmitir información adecuada. Después debido al límite en la distancia que se permite entre imágenes, cuando se producen oclusiones en un nodo este solamente puede volver a su funcionamiento normal

cuando se encuentra una imagen muy cercana a la última imagen válida. Esta condición resulta casi imposible de cumplir. Es por esto que se plantea utilizar la información de las otras cámaras para poder recuperar la información de un nodo.

Esto se realiza añadiendo una conexión de cada nodo con el nodo central, de tal forma que cada cámara siempre pueda tener una retroalimentación correcta de en donde se encuentra la estructura en dicho momento. Esta realimentación de las demás cámaras permite que se realice una comparación entre la información válida proporcionada por las otras cámaras, con la información que cada nodo sigue calculando. De esta forma cada nodo continúa calculando la posición y ubicación con un vector de blobs ordenado con el algoritmo de orden por tamaño de la imagen y cuando el cálculo se aproxima a menos de 90 mm se actualiza la información de la última imagen válida, y se puede recuperar el algoritmo de seguimiento para este nodo.

4.2.5 Discontinuidades

Como último punto es necesario también considerar los casos en que se producen discontinuidades en el cálculo de los ángulos. Aquí existe un error recurrente que se produce debido al comportamiento natural del sistema el cual está limitado entre 180° a -180 º. Esta discontinuidad es normal y permite evitar que se generen rotaciones de más de 360°. Por este motivo se ha verificado que los diversos algoritmos implementados no identifiquen como errores este tipo de discontinuidades, sobre todo debido a que normalmente en este punto se produce la oclusión del marcador más lejano a la cámara.

Se ha comprobado entonces que, en tanto no existan cambios muy bruscos en el ángulo, los valores de distancias máximas son adecuados para seguir el movimiento normal del helicóptero y cambiar el signo del ángulo de giro en los casos que se presente ya que la imagen válida anterior está en un rango lo suficientemente cercano al siguiente, a pesar de que se trate de imágenes muy posteriores a una oclusión.

5 CONTROL

Una vez que se han aplicado los cambios del sistema de visión artificial, como se explicó en el capítulo 4, se necesita que este sistema se comunique correctamente con Matlab, ya que es con esta herramienta que se desarrollará el control del helicóptero coaxial. Para esto es necesario centralizar la información disponible del sistema.

5.1 Cambios en el nodo central

Con los cambios aplicados anteriormente, las 4 cámaras son capaces de funcionar independientemente unas de otras, y trasmitir datos de posición y orientación a cualquier nodo que se desee subscribir a su tópico. También se explicó anteriormente como para implementar un sistema que permita a una cámara validar la información que se encuentra trasmitiendo, necesita recibir la información de las demás cámaras. Es por esto que resulta conveniente para el sistema que el programa dentro del nodo central se encargue de las siguientes acciones:

- Recibir Información de todas las cámaras disponibles
- Validar si la información enviada por las cámaras es fiable
- Calcular el promedio de los valores de posición y orientación del marcador.
- Trasmitir la información correcta, tanto a los nodos de Matlab como al nodo de la cámara que lo necesite

Estas acciones se pueden realizar, tanto modificando el programa del nodo central en la red ROS, en donde se cambien o añadan líneas de programa correspondiente. O a su vez, dentro del mismo Matlab dentro de un subprograma de Simulink. Se escogió realizarlo directamente sobre el nodo de ROS, de forma que el ordenador que trabaje con Matlab, no deba realizar cálculos adicionales con la información, sino que sólo reciba la información correcta, realice los cálculos necesarios para el control y escriba las acciones de control sobre el actuador.

5.2 Comunicación Matlab Control Remoto

Una vez que se ha modificado este nodo central se garantiza que la información de la posición y orientación que recibe Matlab, ha sido previamente validada, y por tanto posee menos ruido y problemas de discontinuidades que podrían afectar al control. Entonces es necesario que

después de realizar los cálculos del controlador, las variables de salida sean escritas en el elemento de control.

En nuestro caso el elemento de control se trata de un control remoto a baterías, modelo WK-241 el cual ha sido modificado físicamente para que se pueda escribir directamente los valores de las consignas para el helicóptero coaxial, mediante el uso de salidas analógicas conectadas a los diferentes potenciómetros de los Joystick.

Estas variables de salida son 4 diferentes y se las denomina:

- Colectivo
- Antipar
- Eje Longitudinal
- Eje Lateral

El colectivo es la variable que se encarga de controlar la velocidad de los motores del helicóptero, o dicho en términos aeronáuticos, controla el empuje, llamado “thrust” en inglés, del helicóptero coaxial.

Esta velocidad de los motores es la que genera la fuerza de sustentación en el helicóptero, la cual depende de la velocidad de giro de ambos motores, y permite cambiar la altura de vuelo.

El antipar en cambio es la variable que se encarga de controlar la rotación del helicóptero sobre el eje de rotación de sus motores. Dicho en términos aeronáuticos controla la dirección o ángulo yaw del helicóptero. Esta variable trabaja sobre uno de los rotores cambiando la velocidad entre ellos, ocasionando que uno de los 2 motores gire a una mayor velocidad y por tanto produzca un mayor par que el otro, y de esta forma el helicóptero puede rotar.

Después el eje longitudinal, se refiere a un plato cíclico el cual forma parte del helicóptero. Este plato se sitúa en el motor inferior cambiando su ángulo de inclinación y también la de todo el helicóptero. Es el mecanismo que permite a los helicópteros inclinar lateral y longitudinalmente el plano de rotación de las palas a la vez que va cambiando el ángulo de ataque estas (en avance tiene que ser mayor que en retroceso). Esta variable trabaja directamente sobre el ángulo de elevación o también llamado ángulo pitch.

Finalmente, el eje lateral, también se refiere al plato cíclico, pero se encarga en cambio de modificar la inclinación a 90° con respecto al eje longitudinal. Esto se refiere al ángulo de alabeo o también llamado roll.

Estas mismas variables se corresponden con las acciones de control que se pueden cambiar mediante el Joystick del control remoto.

Para poder modificar estos valores, se utiliza un swtich que conecta con 4 salidas analógicas para cada canal respectivamente. Estas 4 salidas analógicas se obtienen de tarjetas conversores DAC, las cuales se han instalado en el ordenador número 8.

Se van a utilizar 2 tarjetas DAC, debido a que comúnmente cada tarjeta solo posee 2 salidas analógicas, de las 4 que se necesitan. La primera tarjeta es una tarjeta Advantech PCI, modelo 1710U, la cual se encuentra instalada dentro del CPU del ordenador número 8 del laboratorio de control avanzado. La segunda tarjeta es una tarjeta portable “National Instruments” modelo USB-6008, la cual se conecta al ordenador a través de un cable usb / mini usb.

Las tarjetas son diferentes porque la primera es propia del laboratorio, es decir que todos los ordenadores del laboratorio poseen una tarjeta de este tipo. Mientras que la segunda es una tarjeta que se utiliza debido a que los buses de expansión del ordenador ya están llenos.

Estas se van a conectar de acuerdo al esquema de la figura 27 que se observa a continuación.

Adicionalmente se van a utilizar la tarjeta Analógicas de entrada de la tarjeta Advantech, para poder conocer el estado de las diferentes variables al momento de comenzar el experimento de control, ya que, dependiendo del estado de carga de la batería, estas variables varían de un experimento a otro.

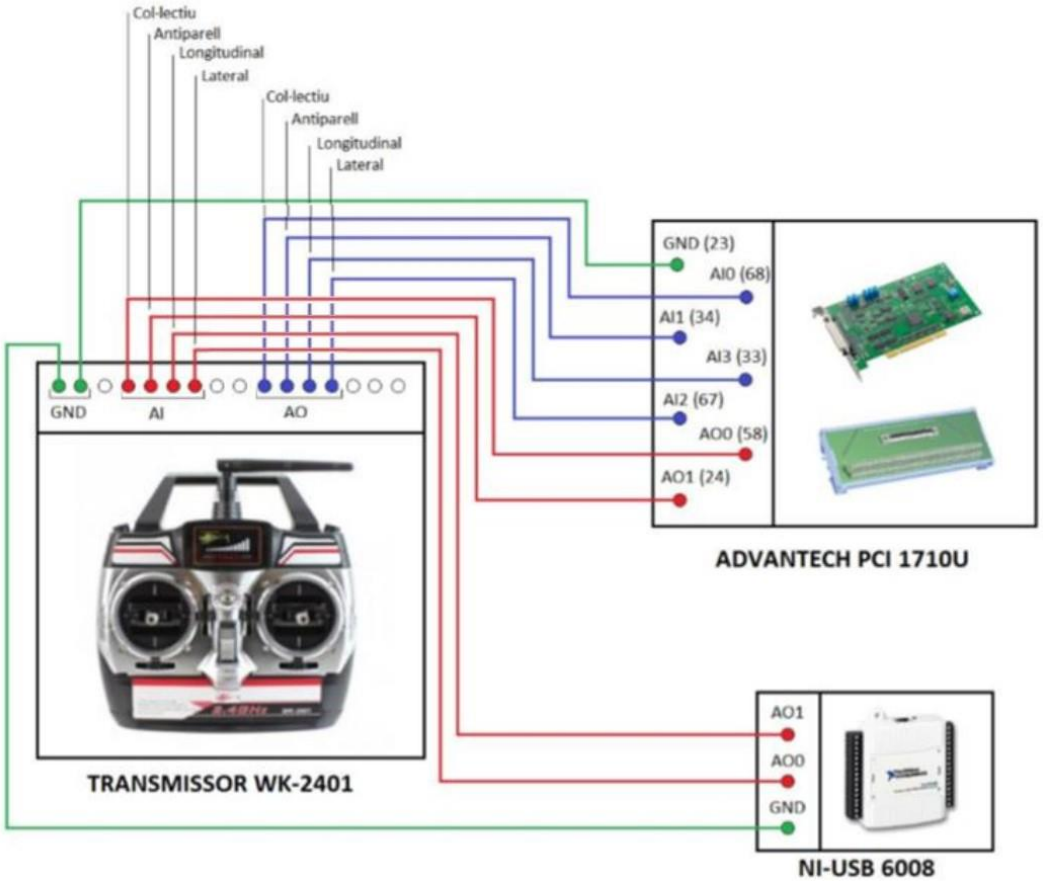

Figura 27 Esquema de Conexiones del control Remoto

5.3 Arquitectura de Control

De esta forma una vez que han quedado definidos los elementos físicos necesarios para poder realizar un control en lazo cerrado, se presenta la arquitectura de control necesaria para el helicóptero la cual se puede observar en la figura 28.

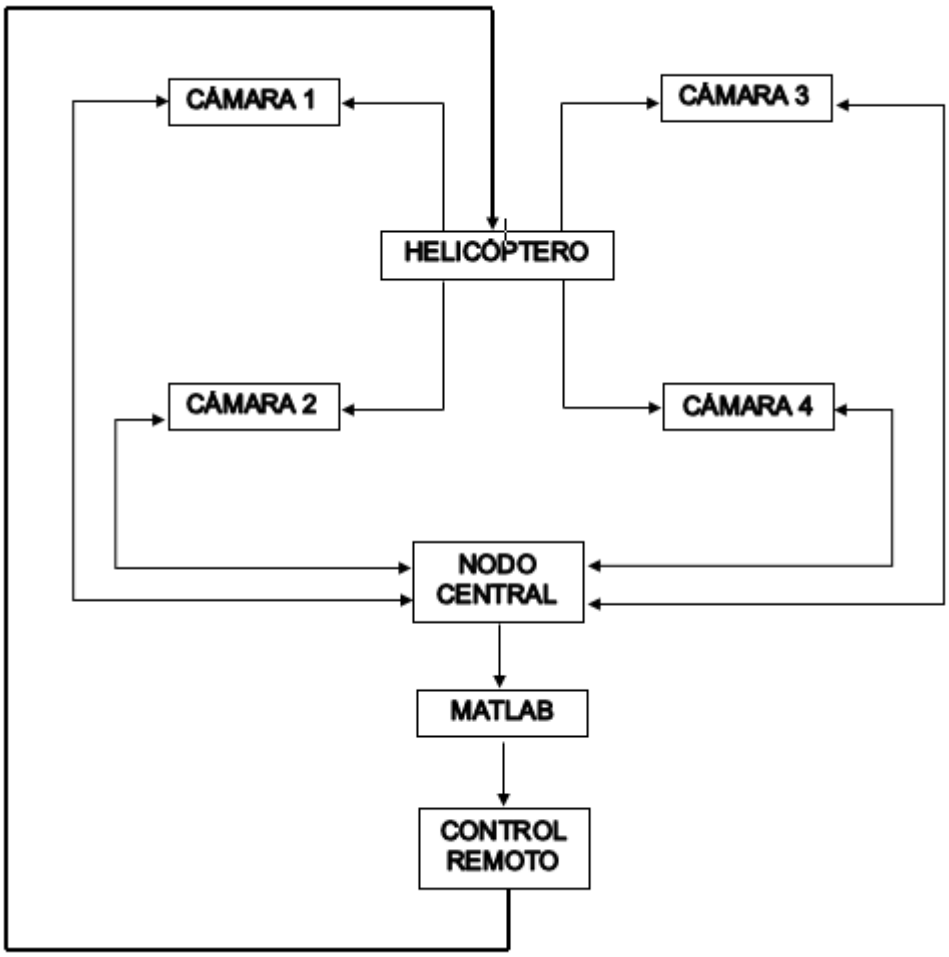

Figura 28 Esquema de control

En esta imagen se puede observar que las cámaras son los sensores del sistema, las cuales reciben imágenes del helicóptero en el laboratorio de control avanzado, y mediante el programa de visión artificial, que posee cada una, determinan unos vectores de posición y orientación del helicóptero. La información de estas 4 cámaras es recibida por el nodo central el cual se ha encargado también de enviar la orden de inicio simultáneo a todas las cámaras, de forma que todas las cámaras disponibles puedan determinar los vectores iniciales en una imagen estática y con los marcadores visibles.

Este nodo central se encarga de recibir la información de las cámaras disponibles, y descartar la información que se encuentre etiquetada como incorrecta, para después calcular el promedio únicamente en los valores restantes. Esta información validada se publica del nodo central en un tópico llamado principal, el cual trasmite mensajes tipo TwistStamped.

Estos mensajes son recibidos tanto en el nodo de ros en Matlab, como en los mismos nodos de las cámaras, para que estas puedan validar sus imágenes y cálculos en los casos en que se encuentren perdidos, y una vez que han sido recibidas en el nodo de Matlab, se calcula a través de los diferentes lazos de control, el valor de salida para el actuador.

Este valor de salida, se escribe mediante los conversores digital/análogo, en el control remoto, el cual finalmente envía estas órdenes al helicóptero a través de radiofrecuencia, en donde son leídas y cambian la tensión en los diferentes motores y servomotores.

5.4 Tiempos de Ejecución de ROS

En el nodo de Matlab es importante entonces comprobar la velocidad de trasmisión de los datos de posición y orientación, para garantizar que no ocasionen problemas con los parámetros de control en Matlab.

Esto se debe a que el control se desarrolla a partir de un esquema de control previamente diseñado, sobre el que se realizaran todas las modificaciones necesarias para su funcionamiento en conjunto con ROS.

Este esquema de control desarrollado, ha sido diseñado para trabajar a una frecuencia de muestreo de 60 hz, que corresponde a la máxima velocidad de muestreo de las cámaras que empleaba como sensor. Entonces se va a comprobar que se mantenga este valor mediante el comando de Matlab, Tic toc.

Esto es un temporizador que permite conocer el tiempo transcurrido entre diferentes acciones. Con este comando se van a medir 500 acciones de recepción de datos, en el nodo ROS de Matlab, para conocer cuál es la periodicidad con que el nodo central trasmite datos.

En la figura 29 se puede observar los tiempos empleados entre una acción consecutiva de recibir un dato publicado, y se observa que la media se encuentra en 33 ms, es decir en 30hz.

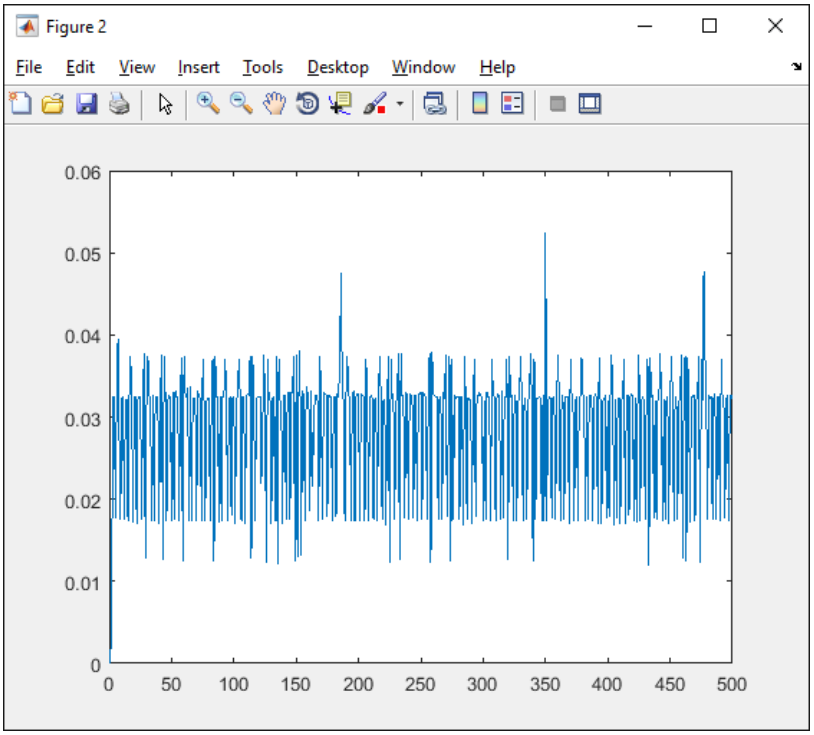

Figura 29 Periodicidad entre datos publicados

Existen sin embargo muchas ocasiones en que el tiempo entre dato y dato se reduce hasta los 16ms es decir la frecuencia deseada de 60hz.

Esto se debe a que la cámara no funciona a una frecuencia fija, sino que dependiendo de la escena que está capturando, puede tardar más o menos tiempo en comprimir la información.

Será necesario entonces comprobar el funcionamiento de los bloques de lectura de simulink, ante estas diferentes velocidades de recepción de datos.

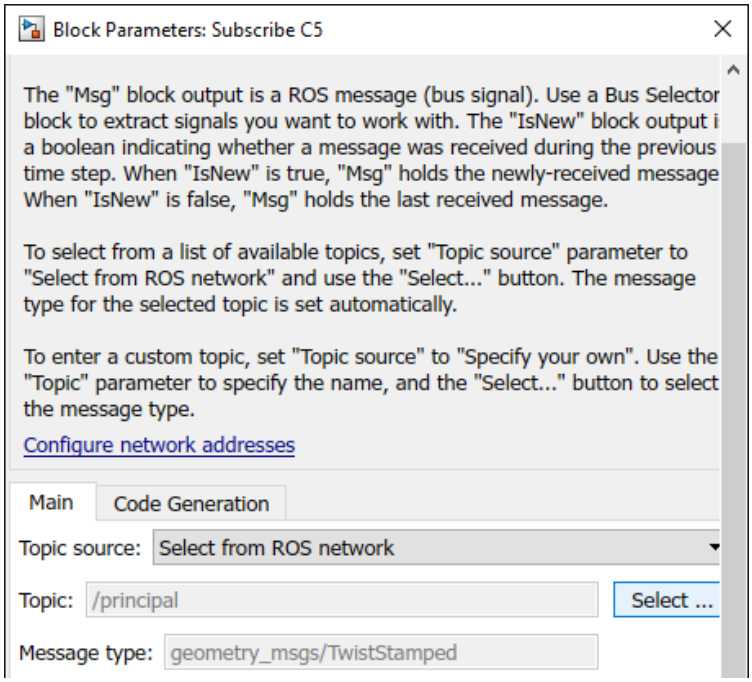

Figura 30 Bloque Subscriptor ROS Simulink

Como se puede observar en la figura 30, este bloquee posee una salida adicional para indicar si el mensaje que se encuentra en cola es nuevo o no. También indica que cada vez que este bloque es invocado, toma el último dato disponible en el tópico al que está suscrito.

De esta manera al no tener un buffer de datos, no existe el riesgo de que los datos se empiecen a acumular si no son leídos, y puedan saturar el tamaño del buffer, o entorpecer la comunicación al requerir acciones de lectura.

Es decir que sí puede trabajar a una velocidad de actualización de 60 hz, ya que no será necesario leer todos los mensajes anteriores, para poder acceder al último dato. El sistema de control siempre dispondrá de un dato válido cada vez que el tiempo de muestreo así lo necesite.

5.5 Esquema de control simulado

Se va entonces a partir de un esquema de control que se ha probado previamente con un modelo matemático del helicóptero coaxial. Este esquema se puede observar en la figura 31

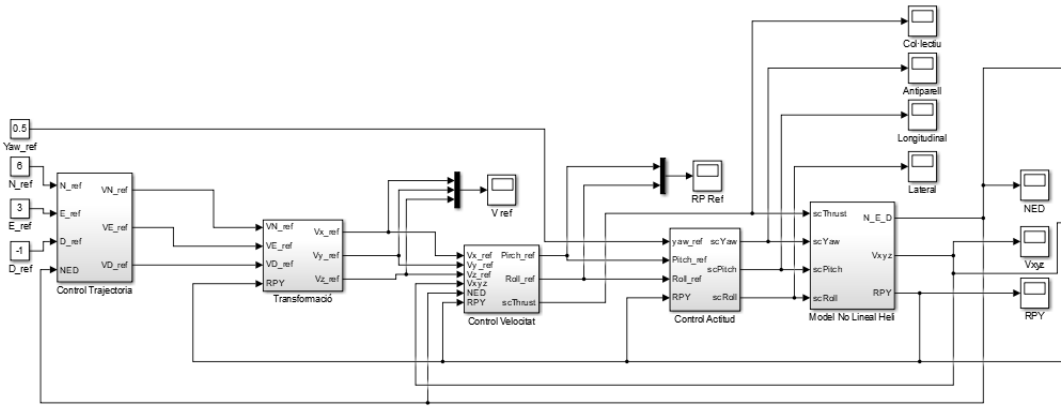

Figura 31 Esquema de control base

En la parte lateral derecha tenemos el modelo del helicóptero coaxial como un sistema no lineal, al cual ingresan las variables del colectivo, antipar, plato longitudinal y plato lateral, y nos retorna la posición en coordenadas NED, junto con una estimación de la velocidad y la orientación en ángulos Pitch, Roll y Yaw.

Las coordenadas NED, se refieren a un sistema de coordenadas geográficas, usadas comúnmente en aviación, y se refieren a la posición a lo largo del eje Norte(North), a lo largo del Este (East)y a lo largo del eje Bajada (Down), para cumplir con la regla de la mano derecha. Se utilizan estas coordenadas para tener un sistema de coordenadas fijo, en vez de las coordenadas x,y,z que varían de acuerdo a la orientación del helicóptero.

Antes del modelo del helicóptero se observa un bloque en donde se realiza el lazo de control de la actitud del helicóptero, es decir su comportamiento. Este bloque se observa en detalle en la figura 32 y está compuesto por 3 controladores PID independientes, que han sido sintonizados para mantener las consignas de los variables Pitch, roll y yaw.

Estas variables se deben controlar en el lazo más interno debido a que son las variables que determinan la estabilidad del helicóptero. Si alguno de estos ángulos evoluciona alejándose de los puntos estables, la dinámica del sistema ocasionará una gran inestabilidad que afectará también a los lazos más externos.

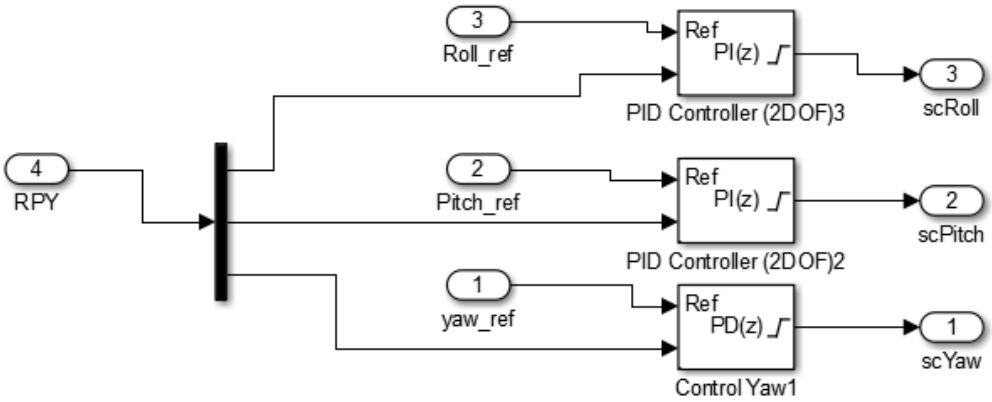

Figura 32 Bloque de Control de Altitud

Por fuera de este lazo de control de la actitud del helicóptero, se observa un segundo lazo de control de la velocidad del helicóptero. Este lazo de control de velocidad se observa en detalle en la figura 33, donde se observan 3 controladores PID independientes.

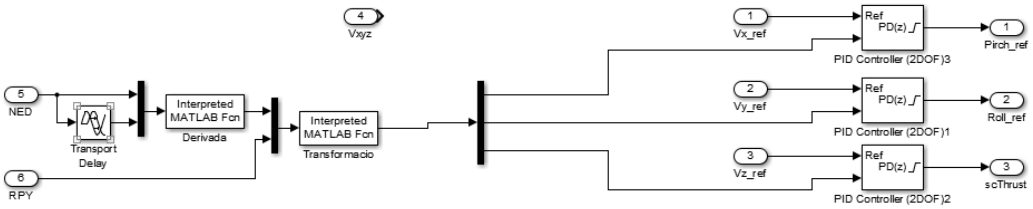

Figura 33 Bloque de Control de Velocidad

El primer lazo está sintonizado para mantener una consigna de velocidad sobre el eje x, por lo cual requiere al sistema un cambio en la consigna de ángulo pitch, es decir el ángulo de elevación que permite que el helicóptero se desplace axialmente sobre su eje longitudinal.

El segundo lazo es similar al lazo anterior ya que está sintonizado para mantener una consigna de velocidad sobre el eje y, y por tanto requiere al sistema un cambio en la consigna del ángulo roll, el cual es el ángulo de alabeo que permite al helicóptero desplazarse lateralmente.

El tercer lazo se sintoniza también para mantener una consigna de velocidad sobre el eje z, requiriendo un cambio en la consigna del empuje del helicóptero para controlar la fuerza de sustentación que permite al helicóptero ascender o descender.

Queda por fuera de este bloque de control la consigna del ángulo yaw, es decir la rotación del helicóptero sobre el eje z, la cual debe ser establecida por el usuario en un valor determinado, o a su vez se debe tomar el valor en que se encuentre al momento de realizar el experimento.

También se realiza dentro de este bloque de control, la estimación de la velocidad del helicóptero a través de su posición. Esta estimación es una estimación con retraso ya que se calcula de acuerdo a la fórmula:

$$V = \frac{Pos_{final} - Pos_{inicial}}{\Delta tiempo}$$

Donde se necesitan 2 puntos diferentes para estimar la velocidad, y tiene la desventaja que esta fórmula no toma en cuenta los diferentes factores dinámicos del modelo para establecer con una mayor exactitud la evolución de la velocidad del helicóptero.

Finalmente, antes del lazo de control de velocidad existe un bloque de control encargado de realizar el control de la Posición y se puede observar en detalle en la figura 34.

Aquí se observa que la posición actual del helicóptero ingresa al sistema a través del vector NED, que contiene los valores de posición actuales para ser comparado con unos valores de posición de referencia, los cuales son los establecidos por el usuario, o los que se generan en el control de la trayectoria. Las salidas de este bloque de control son las velocidades en ejes inerciales, que servirán de referencia necesarias para los lazos de control internos.

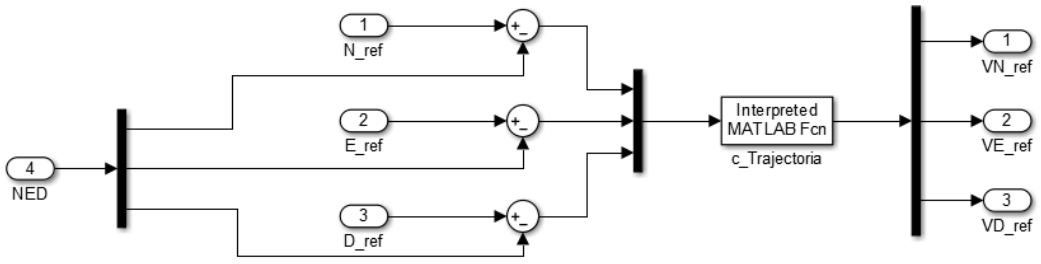

Figura 34 Bloque de control de la posición

5.6 Esquema de control aplicable

El esquema de control anteriormente explicado presenta una arquitectura clara y evidente que se puede utilizar para realizar el control del helicóptero de una manera correcta, sin embargo, presenta también elementos que no

se han explicado en el algoritmo de control. Uno de estos elementos son las transformaciones del sistema de coordenadas fijo o mundo, respecto al sistema de coordenadas inercial NED.

El sistema de coordenadas mundo, se trata de la convención de planos y ejes comúnmente utilizados, y comúnmente conocidos como plano cartesiano. Es decir el sistema de coordenadas x,y,z que se ha establecido para el laboratorio de control avanzado y que se corresponde con las coordenadas del sistema de visión artificial.

Por otra parte, el sistema de coordenadas inerciales NED, es un sistema de coordenadas inercial en la cual no se toman en cuenta las fuerzas ficticias, producidas por el movimiento del helicóptero y por tanto se corresponde con las coordenadas vistas desde la perspectiva del helicóptero. Ambos sistemas son diferentes, por lo que es necesario realizar transformaciones entre ellos.

Finalmente es necesario también incluir en el sistema de control un método para indicar el comienzo del experimento, y también las condiciones de vuelo del helicóptero en ese momento. Es decir que el experimento comenzará después de que el helicóptero haya despegado manualmente y haya sido llevado a una posición de Hoover. Una vez allí es necesario que la información de la posición en x,y,z sea almacenada como consigna del experimento, junto con la consigna de ángulo yaw, como se explicó anteriormente.

Para poder realizar esta condición se ha añadido dentro del modelo real del helicóptero los siguientes bloques como se ven en la figura 35.

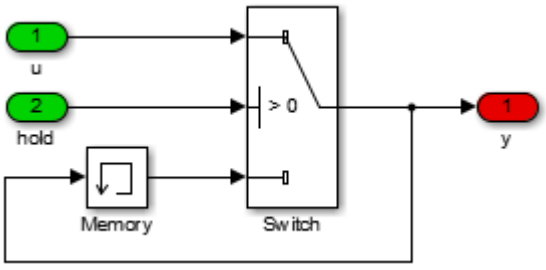

Figura 35 Estructura de bloques para almacenar un valor

Aquí se observa que para almacenar el valor de la variable u se emplea un selector y una memoria conectada de la salida del sistema. De esta forma

la salida del sistema se mantiene en el último valor de la memoria cuando la señal de hold es menor que cero.

Con esta arquitectura se puede almacenar los valores actuales de posición como una referencia constante, que será la referencia a mantener por el sistema de control, y permite también utilizar este cambio de flanco del selector, como un disparador para resetear la parte integral de los diferentes controles PID.

6 PRUEBAS

En este capítulo se va a explicar las diferentes pruebas que se han realizado con el helicóptero coaxial, tanto a nivel físico como a nivel de programación en ROS, Matlab y simulink. Debido a la naturaleza inestable y muy acoplada del sistema, no se pueden probar los diferentes lazos de forma independiente, sino que se debe trabajar todo en conjunto. Esto supone una mayor dificultad al momento de realizar experimentos debido a que todos los elementos antes explicados deben estar en óptimas condiciones para poder realizar las pruebas correctamente.

Las primeras pruebas entonces que deben realizarse serán pruebas de tipo físico para comprobar el buen estado de todos los elementos físicos del sistema, desde las baterías hasta el mismo helicóptero.

6.1 Pruebas de elementos físicos

La fuerza de sustentación del helicóptero depende entre otras cosas de la velocidad de rotación de los motores. Estos motores son motores de tipo DC, es decir que su potencia y su velocidad de giro, dependen de la intensidad de corriente absorbida. Esta intensidad de corriente la obtienen de las baterías que transportan por lo cual estas deben encontrarse en buen estado.

Se disponen en el laboratorio de diferentes baterías Li-Po, es decir de Litio y polímero, de 3,7 V, con una capacidad desde los 900mAh hasta 1200mAh, de diferentes marcas y con un peso aproximado de 27 gramos, de las cuales se observan los diferentes tipos disponibles en la figura 36.

Figura 36 Diferentes Baterías Li-Po

De estas baterías de diferentes tipos existen en total 15 diferentes baterías que se puede emplear en el helicóptero, sin embargo, se ha determinado experimentalmente que solo 7 baterías en total son capaces de funcionar adecuadamente, mientras que las demás ya no son capaces de entregar la misma cantidad de corriente eléctrica, o se descargan muy rápidamente. Cada una de estas baterías tiene un efecto diferente sobre el helicóptero, ya que física y eléctricamente son también diferentes entre sí. Por este motivo es importante que en cada vuelo que se realice, se conozca el valor del canal de Empuje, thrust, en el Joystick, ya que dependiendo del número de experimentos previos y también de la batería empleada, este valor cambiará entre cada experimento.

Para esto se van a emplear las señales de las entradas digitales, obteniendo el dato de los diferentes potenciómetros, para de esta forma, conocer la altura de referencia en cada experimento y ajustarla a la altura en que el helicóptero haya podido realizar un hover en ese momento.

Adicionalmente, al revisar el estado del helicóptero coaxial, se ha encontrado que existe un rozamiento entre los engranes del motor inferior con la base del mismo, lo que además de un ligero ruido ocasiona una pérdida de potencia por rozamiento y consecuentemente una pérdida de altura de vuelo. Esto se ha corregido mecánicamente ajustando el juego que existía entre el eje del motor y la parte superior del mismo. También se ha notado que la barra estabilizadora del helicóptero se encontraba deformada como se observa en la figura 37, motivo por el cual se la ha reemplazado con una nueva.

Figura 37 Barra Estabilizadora Deformada

6.2 Pruebas de vuelo del Helicóptero coaxial

Una vez que se ha comprobado el funcionamiento de las baterías se ha decidido probar el helicóptero coaxial pilotado desde el control remoto. Para esto se ha modificado la estructura del helicóptero retirando algunos elementos de los cuales se puede prescindir. La estructura inicial del helicóptero se observa en la figura 38.

Figura 38 Estructura Inicial del Helicóptero

De esta estructura se va a retirar toda la carcasa exterior, las barras que sirven como tren de aterrizaje y la barra posterior en donde va la cola del helicóptero, dejando únicamente la estructura esencial para los motores como se puede observar en la figura 39.

Figura 39 Estructura básica del Helicóptero

Una vez que se han retirado todos los elementos posibles, sin comprometer el adecuado funcionamiento del helicóptero, es necesario montar la estructura de marcadores y adicionalmente un tren de aterrizaje. Este tren de aterrizaje se observa en la figura 40, y es muy importante en el helicóptero, ya que, en los diversos experimentos, la fuerza de sustentación

cambiará constantemente, por lo cual se necesitaría de mucha experiencia, para evitar aterrizajes bruscos que puedan dañar el helicóptero.

Figura 40 Tren de Aterrizaje

Con estas modificaciones se ha realizado una prueba de vuelo con el control remoto, sin embargo, este no ha alcanzado una altura considerable, por lo cual será necesario realizar más cambios físicos.

6.3 Cambios físicos

El helicóptero coaxial, se encuentra muy limitado respecto a la carga que puede elevar, ya que los motores que lleva le permiten únicamente elevar su propio peso. Por este motivo se va a quitar el tren de aterrizaje y se sustituye por piezas de poliestireno expandido, las cuales presentan un peso muy bajo y son capaces de absorber los golpes a alta velocidad.

Esta solución no consiguió mejorar significativamente el resultado, debido a que la diferencia en pesos entre ambos materiales es muy pequeña, lo cual parece indicar que la falta de elevación no se debe únicamente al exceso de peso de carga, sino a una mala distribución de esta que traslada el centro de masas fuera del eje de rotación. Este problema resulta significativo ya que cuando el centro de masas no se encuentra cercano al eje de rotación de los motores, ocasiona un desequilibrio que debe ser corregido cambiando el ángulo del plato cíclico bien sea longitudinal o transversalmente. Este cambio del ángulo para compensar el desequilibrio, resta fuerza de elevación al helicóptero.

Por esta razón se han probado las posibles ubicaciones de los marcadores en que las barras coinciden con los ejes longitudinales y transversales del helicóptero, sin embargo, ninguna de ellas ha conseguido disminuir este

efecto. Entonces se ha cambiado la orientación de los marcadores de forma que las barras no coinciden con los ejes del helicóptero, sino que se encuentren a 45° de estos. Esta configuración se observa en la figura 41.

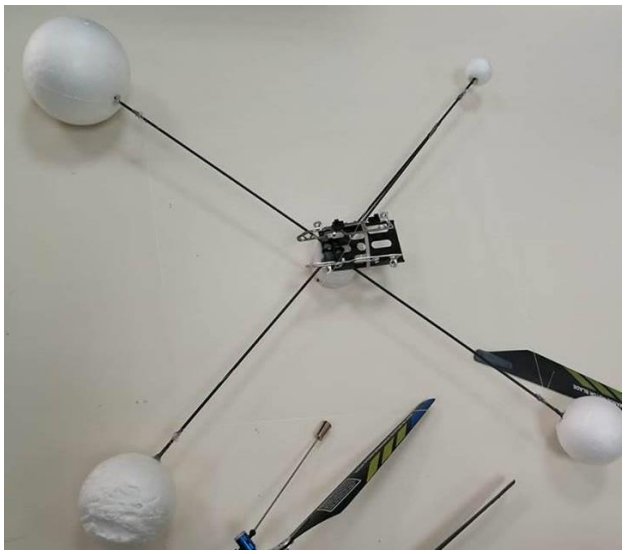

Figura 41 Nueva Distribución de los marcadores

Esta distribución sí consigue acercar la ubicación del centro de masas al eje de rotación de los motores, además que permite utilizar los mismos marcadores para la visión artificial, como tren de aterrizaje del helicóptero permitiendo retirar también las barras de poliestireno expandido.

A pesar de las ventajas, aparece un nuevo problema debido a que la estructura está en contacto directo con la base de los motores del helicóptero. Esto ocasiona que la vibración natural de los motores, se traslade a las barras y ocasione una vibración muy notoria en el marcador con la bola más pequeña. Esta vibración, ocasiona problemas con el sistema de visión artificial, por lo cual se lo busca corregir cambiando el tipo de barras, que son delgadas y sólidas, por unas barras de fibra de carbono, con forma rectangular huecas. Esta forma resulta más rígida que las barras delgadas, por lo cual reduce la vibración, pero no lo suficiente para el sistema de visión.

Entonces se va a descartar esta idea y retomar las barras delgadas añadiendo una barra delgada a la barra del marcador más pequeño, de forma que esta realice una fuerza contraria al desplazamiento del marcador por vibración, y reduzca así la vibración. Esta solución consigue disminuir la vibración vertical del marcador, pero no las vibraciones laterales de este.

Finalmente se van a añadir a la estructura, hilos de nylon, debido a su bajo peso y alta resistencia a la tensión, los cuales van a atravesar las diferentes barras de los marcadores y sujetarse sobre ellos para mantener la posición de las barras tensionadas, reduciendo de esa forma las vibraciones laterales del marcador más pequeño. Esto se observa en la figura 42.

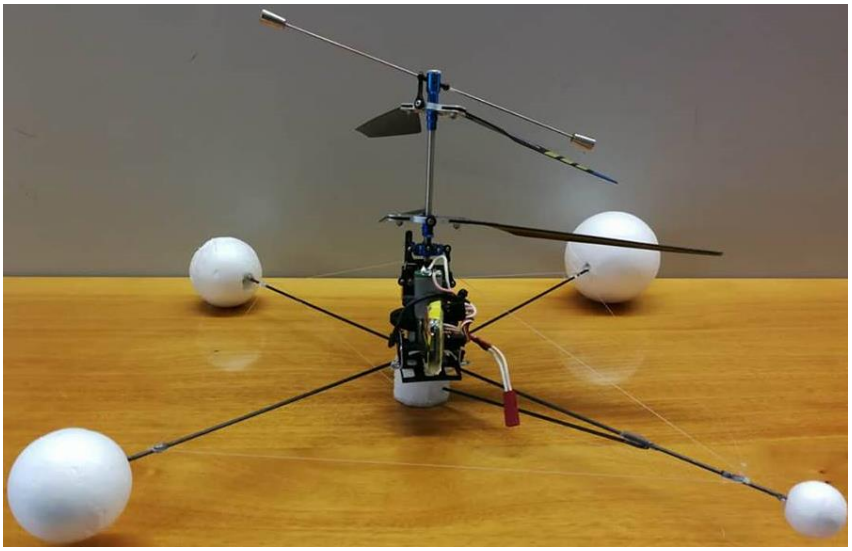

Figura 42 Barras Adicionales e Hilos Nylon

Con todos estos cambios en el helicóptero se ha conseguido mejorar la altura de elevación, sin embargo, no se ha mejorado el tiempo de vuelo del helicóptero el cual es menor a los 3 minutos. Por este motivo se decidió cambiar la fuente de alimentación del helicoptero de baterías a una fuente variable de corriente continua.

Esta solución tiene la ventaja de que una fuente de variable de corriente continua no se agota después de cierto tiempo, sino que permite trabajar por largos periodos siempre y cuando no se sobrecalienten los motores o cables. Adicionalmente reduce la carga del helicóptero al quitar el peso de la batería del sistema. Sin embargo añade un nuevo peso, debido a los cables que van desde la fuente de corriente continua al helicoptero, y tambien limita considerablemente el espacio de vuelo del mismo. Otro problema importante se encuentra en la seguridad, ya que si se producen giros bruscos en el helicoptero el cable puede enredarse y ocasionar daños al helicoptero.

Tomando las precauciones necesarias se ha conectado el helicóptero a diferentes fuentes variables de corriente continua como se observa en la figura 43, y partiendo de una fuente de 1,5 Amperios se ha aumentado

gradualmente la corriente máxima que ingresa al helicóptero hasta llegar a 4 Amperios, que es el valor máximo que se obtuvo trabajando con una fuente paralela. Sin embargo estos valores de corriente no son suficientes para que el helicoptero pueda despegar del suelo, y cuando este trata de absorver más corriente de la fuente, todas las fuentes se apagan debido a sus protecciones contra cortocircuitos.

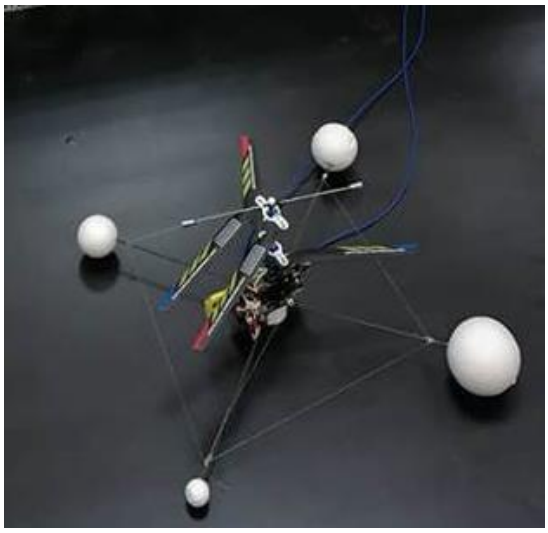

Figura 43 Helicóptero conectado a Fuentes DC

En vista de que no es viable realizar este cambio se debe retornar a trabajar con baterías y se ha decidido entonces para mejorar el sistema, sustituir los motores con sus respectivos engranajes, con los de otro helicóptero de las mismas características, que se encuentra en el laboratorio como fuente de recambios.

Al momento de hacer este cambio es muy importante mantener la conexión de los nuevos motores de acuerdo a lo que se indica en la tarjeta controladora. Esta tarjeta se puede observar en la figura 44. Si no se mantiene el orden de conexión de los motores, el helicóptero comienza a girar sobre sí mismo sin control, debido a un error en el sentido de las realimentaciones dentro de la controladora. Esta controladora posee integrado un giroscopio, el cual debe mantenerse siempre en posición vertical como se indica en la figura 44.

Figura 44 Tarjeta Controladora

Este giroscopio en la controladora se encarga de medir la orientación del helicóptero respecto al eje de sus motores, es decir el ángulo yaw. Si el helicóptero comienza a girar sobre sí mismo, se debe a que la velocidad de un motor es mayor que la del otro, por lo cual para corregir este error la tarjeta controladora debe igualar la velocidad en ambos motores.

Esta acción la realiza aumentando o disminuyendo la velocidad del motor superior, para igualarla con la del motor inferior. Cuando se han invertido los motores, la controladora tiene entonces una ganancia positiva, la cual aumenta la diferencia entre las velocidades, y aumenta el error a la entrada. Esto a su vez aumenta la acción de control invertida que aumenta más la diferencia, y así sucesivamente hasta llegar al punto de saturación, en donde el helicóptero gira sin control a máxima velocidad.

Una vez que se ha conectado correctamente los nuevos motores con sus respectivos reductores, se ha conseguido una altura de vuelo adecuada la cual se puede mantener por un tiempo mayor a 3 minutos.

6.4 Pruebas de Inicialización

Con estas modificaciones al helicóptero se ha cambiado la posición de los marcadores en el sistema de coordenadas mundo, así como también las distancias entre los centros de los marcadores y el origen de coordenadas, las cuales se habían definido antes en el archivo de configuración del programa de visión artificial. Los antiguos y nuevos valores de ubicación son los siguientes:

Tabla 5 Ubicación marcadores

	Antiguo			Nuevo		
Marcadores	x	y	z	x	y	z
Más grande	0	205	35	-160	-180	40
Más Pequeño	205	0	52	-165	165	25
2do Más Pequeño	-210	0	40	160	150	30
2do Más Grande	0	-210	55	150	-165	35

Con estas nuevas ubicaciones el programa ya será capaz de realizar el cálculo correctamente. Para verificarlo primero se debe revisar que todas las cámaras sean capaces de identificar los 4 blobs de la estructura. Las diferentes imágenes que captura cada cámara se observa en la figura 45 en el siguiente orden:

- Cámara 1: Esquina Superior Izquierda
- Cámara 2: Esquina Superior Derecha
- Cámara 3: Esquina Inferior Izquierda
- Cámara 4: Esquina Inferior Derecha

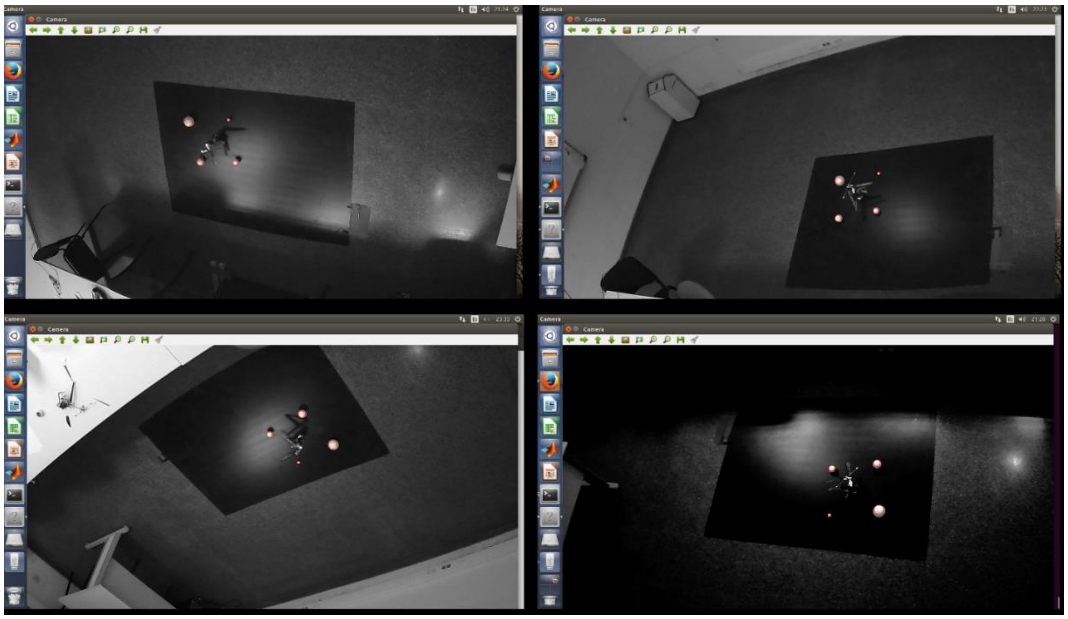

Figura 45 Imágenes recibidas en cada cámara

Aquí se observa que las condiciones de iluminación son las adecuadas ya que en todas las cámaras se han señalado 4 blobs, que corresponden a los 4 marcadores físicos añadidos al helicóptero, y no existen reflejos de luz

que indiquen blobs donde no existen marcadores. Se puede entonces iniciar los programas en cada cámara.

Este proceso se debe realizar en cada ordenador que posee una cámara conectada, lo cual implica que, para iniciar y finalizar el programa de visión artificial, la persona a cargo del experimento debe estarse moviendo constantemente por todo el laboratorio. Para evitar esta situación se ha instalado la aplicación ssh que nos permite acceder a cada ordenador de forma remota y controlar el ordenador desde una terminal, como si esta terminal se estuviera ejecutando en el ordenador.

Con esta aplicación se puede tomar control de los ordenadores: Control 10,7 y 4 que tienen conectados las cámaras 1,2 y 3 respectivamente, e iniciar todas las cámaras desde el ordenador Control 6 en donde está conectada la cámara 4. Esto se puede observar en la figura 46 en donde se ha dado la orden de inicio de cada cámara en una terminal diferente.

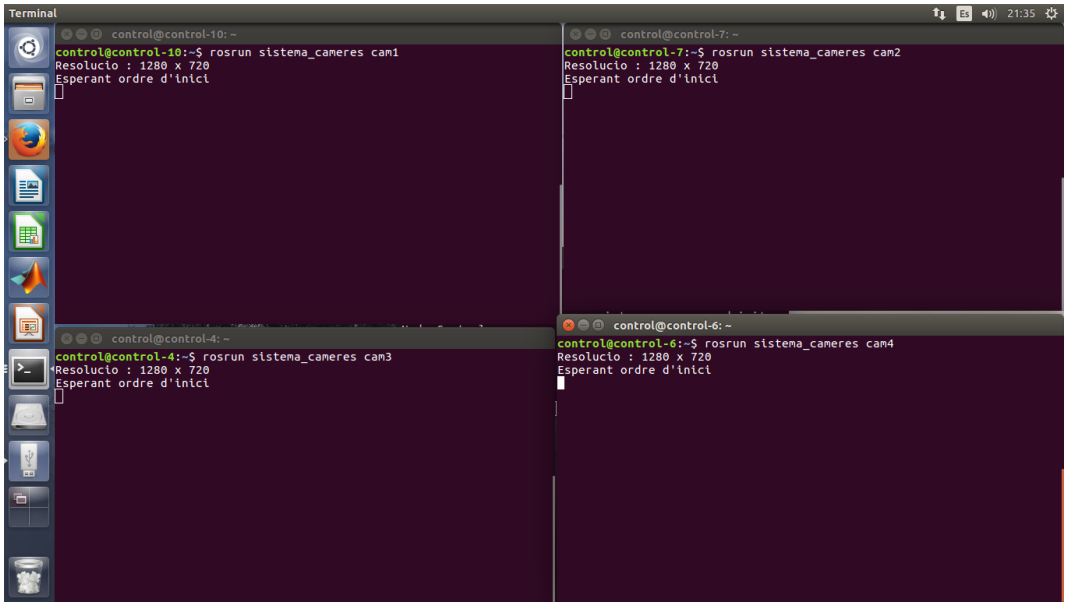

Figura 46 Control Remoto

Dentro del ordenador Control 6 también se encuentran los programas nodo inicial y principal, y al invocar el programa del nodo inicial y escoger el número 1, se observa el siguiente resultado de la figura 47.

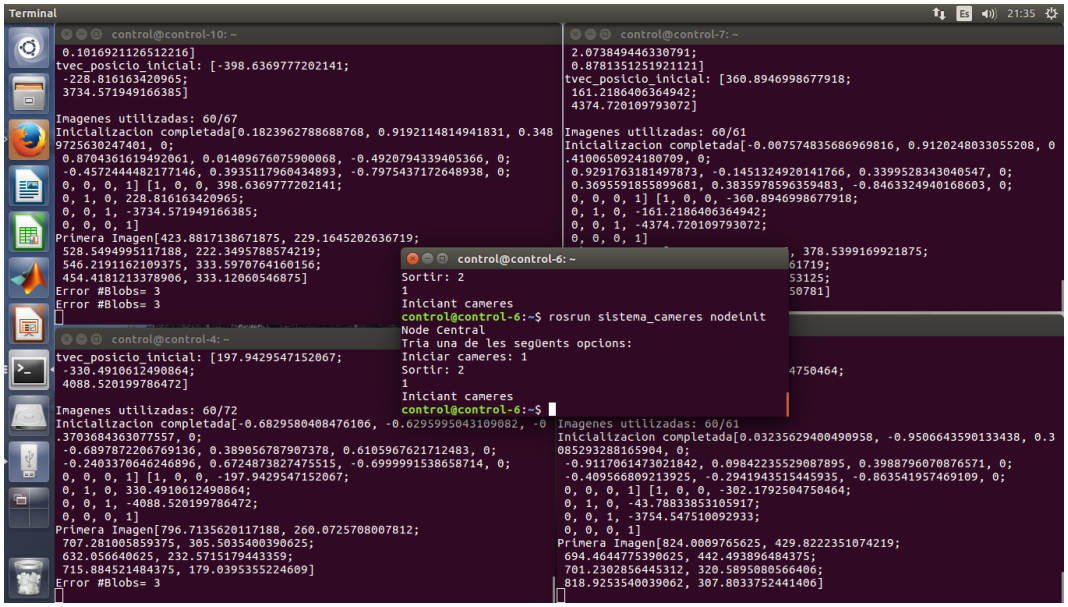

Figura 47 Inicialización Exitosa

En esta figura se puede observar que en cada terminal aparece el número de imágenes utilizadas para inicializar, así como también el mensaje Inicialización Completada, junto con algunos valores de referencia, en caso de que el usuario necesite comprobarlos y son:

- La matriz de rotación inicial
- La matriz de traslación inicial
- La ubicación en coordenadas cámara de los 4 marcadores

Sin embargo, cuando no ocurra una inicialización exitosa de alguna cámara, aparecerá el mensaje “No se ha podido iniciar”, y se cerrará dicho nodo.

El estado de cada cámara se puede monitorizar tanto en la terminal en donde se ejecuta el nodo, o a su vez mediante el comando “Rostopic echo” con el cual se puede mostrar en pantalla toda la información publicada por un tópico en específico. En la figura 48 se muestra el estado de cada cámara, con el mismo orden utilizado para el control remoto, y se comprueba que todas las cámaras están libres de errores ya que se encuentran publicando el mensaje “Ok” junto con la posición en mm y orientación en radianes que han calculado.

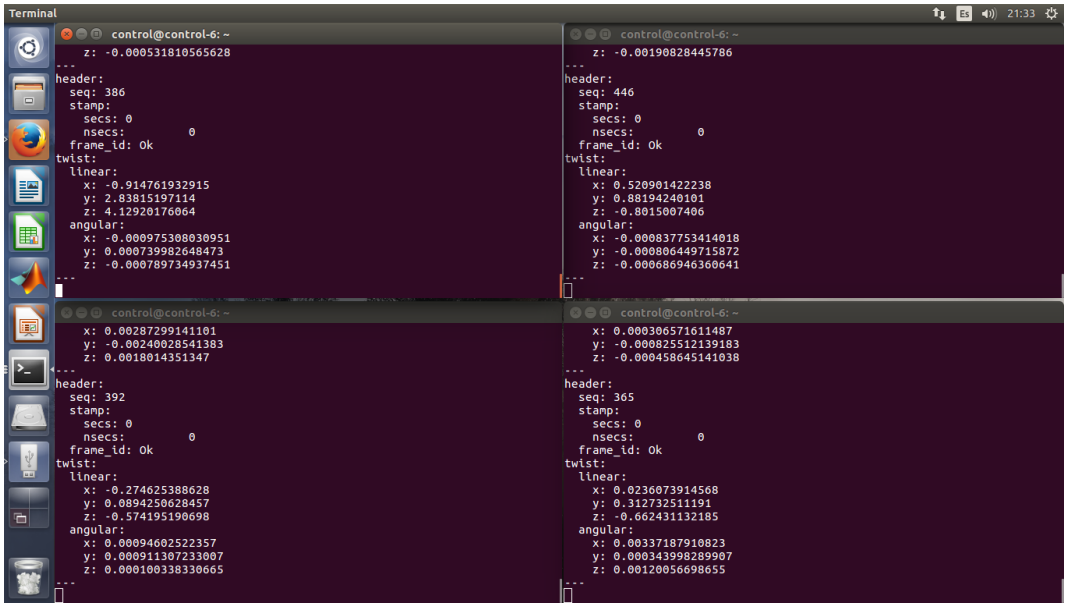

Figura 48 Monitorización de las cámaras

6.5 Pruebas de Seguimiento

Los tópicos de las cámaras publican la información de la posición y orientación en milímetros y radianes respectivamente, sin embargo, estas unidades resultan difíciles de entender, mientras que los centímetros y los grados resultan más fáciles. Por esta razón se va a realizar un cambio de unidades en los datos que se muestran en los gráficos de simulink, en vez de realizar un cambio de unidades directamente sobre el nodo que publica la información, ya que así se mantiene la precisión del dato calculado. Este cambio de unidades se muestra en el siguiente diagrama en simulink en la figura 49.

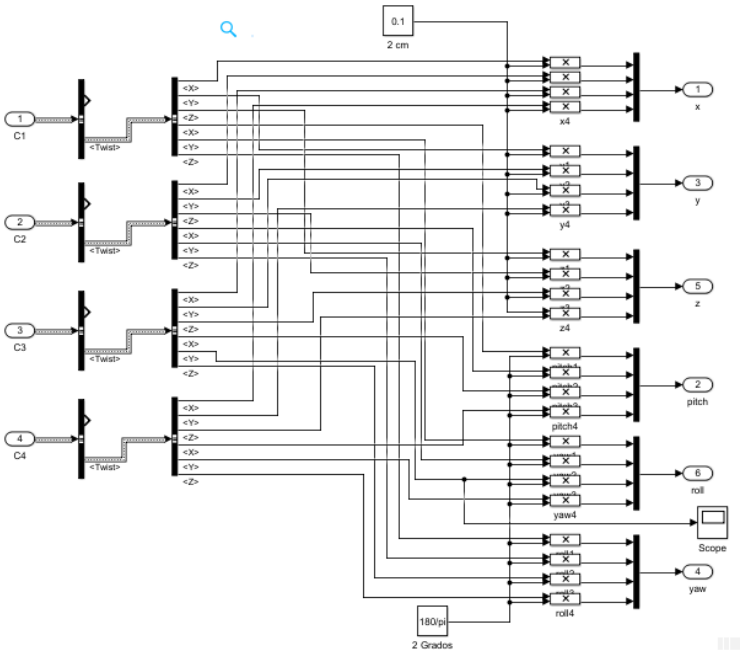

Figura 49 Cambio de Unidades

En esta figura la información de los diferentes mensajes en cada tópico ingresa por un canal y es separada en sus 2 componentes Header y Twist, para después separar el mensaje tipo Twist en sus 6 componentes linear x,y,z y angular x,y,z, como se explicó en el capítulo 2, y una vez que se han separado se realiza la transformación de unidades, para finalmente unirse a la salida de acuerdo al mensaje que trasmiten ordenados del tópico 1 al 4. Estos datos ordenados son los que se van a mostar en un solo gráfico como se muestra en la figura 50.

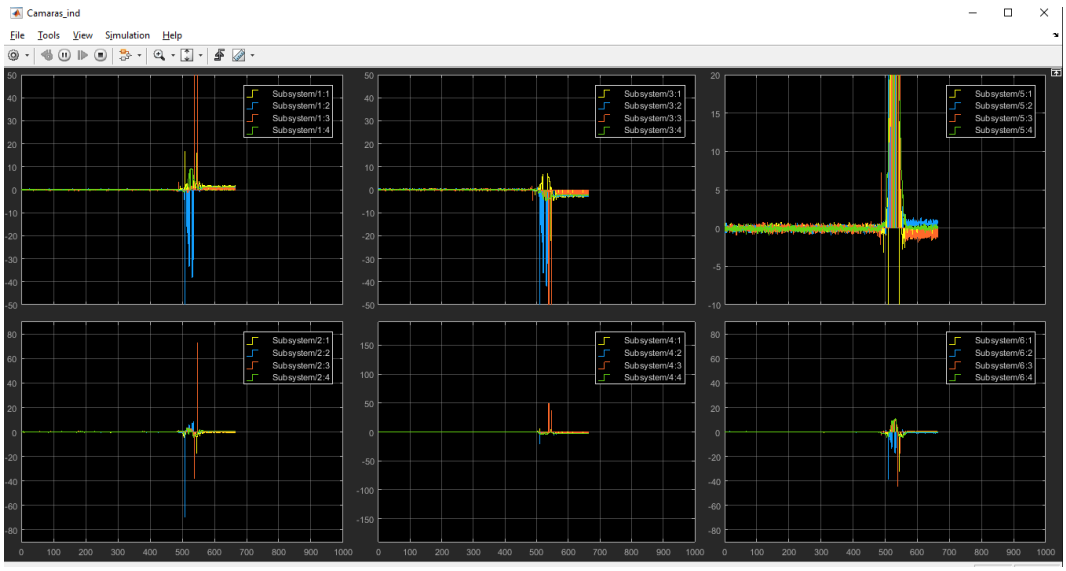

Figura 50 Gráfico Unificado de las Cámaras

La información se encuentra ordenada de la siguiente manera: En la parte superior se muestra la información de la posición, es decir las coordenadas x,y,z en ese orden, mientras que en la parte inferior se muestra la información de la rotación, es decir los ángulo pitch, yaw y roll respectivamente. Además, se muestra en color amarillo la información trasmitida por la cámara 1, en color azul por la cámara 2, en color naranja por la cámara 3 y en color verde por la cámara 4.

Esta misma acción se realiza para el nodo principal, con la diferencia que en el nodo principal sí se ha incluido un cambio de unidades, de milímetros a metros, para que se corresponda con las unidades de trabajo del esquema de control. El resultado se puede observar en la figura 51.

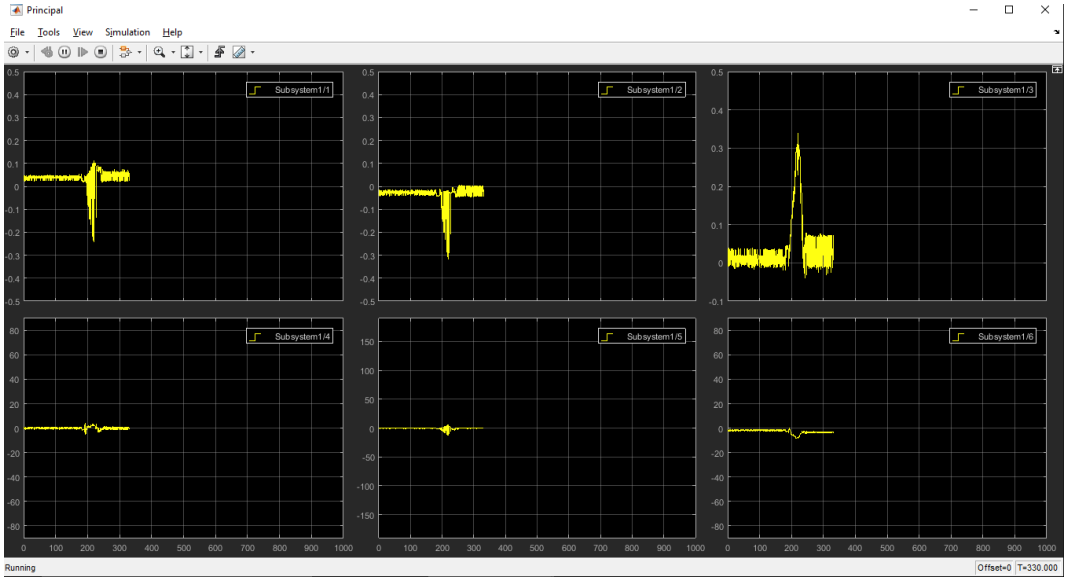

Figura 51 Gráfico Unificado de las Cámaras

En las figuras 50 y 51 se observa también como las cámaras han seguido un movimiento a lo largo del eje z del helicóptero, y a pesar de que la cámara 1, que se observa en color amarillo en la figura 50, ha tenido problemas con el cálculo, estos problemas no han sido tomados en cuenta en el cálculo del promedio que se muestra en el nodo principal en la figura 51.

6.6 Pruebas de Recuperación

Otra prueba que se va a realizar con el helicóptero en movimiento, se refiere a los algoritmos de recuperación cuando ha existido perdida de información. En la figura 52 se observan el momento en que, debido a un error del número de blobs, y al seguimiento que realizan los programas,

luego se produce un error en el que el nodo no puede establecer si los blobs más cercanos se corresponden con el marcador correcto.

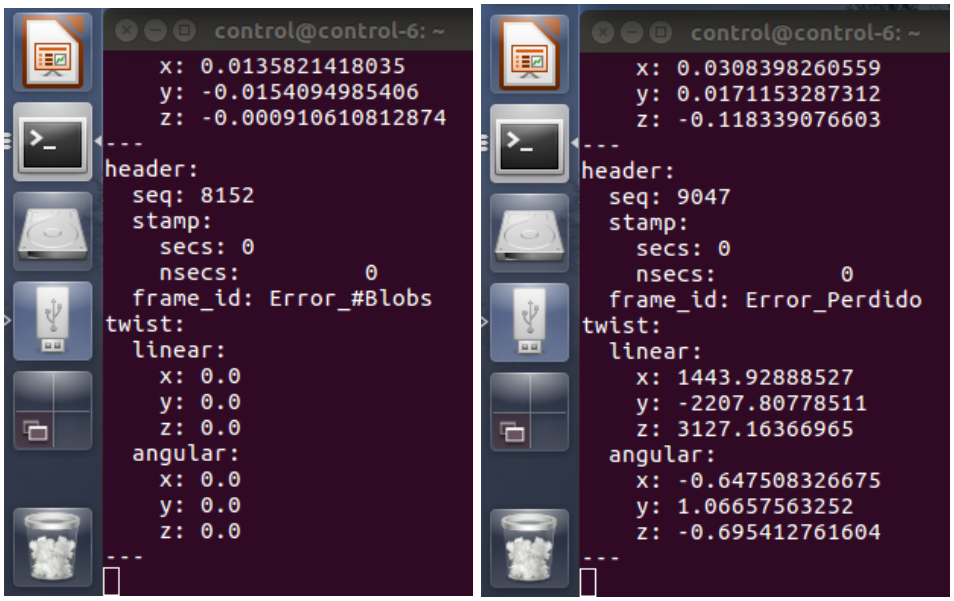

Figura 52 Mensajes de Error

En estos casos se va a mostrar, en la pantalla de la terminal en que se ejecuta el nodo, la posición que se realimenta del nodo principal, junto con la información que ha calculado para ese punto, junto con la distancia total en milímetros que ha calculado entre estos puntos, como se observa en la figura 53.

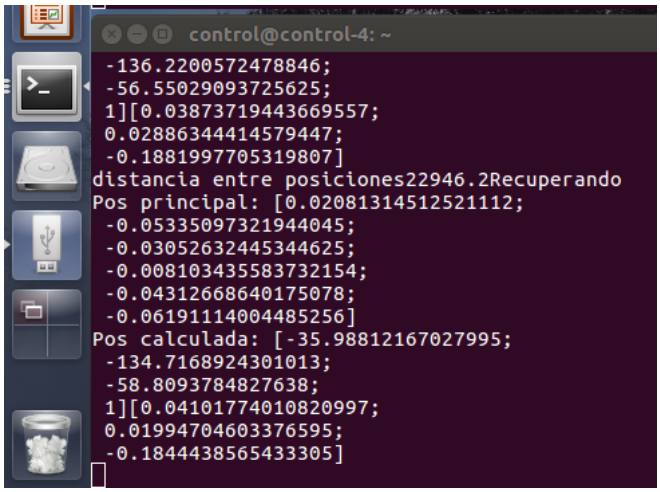

Figura 53 Recuperación de la información

Una vez que se ha recuperado exitosamente la información calculada por una cámara, el mensaje publicado vuelve a ser Ok y se actualizan las últimas coordenadas cámara válidas de los marcadores.

Lo mismo ocurre cuando el error no se trata de la posición calculada, sino de la orientación de los marcadores, con la diferencia de que el mensaje cambia a ser “Error Rvec” refiriéndose a un error en el vector de rotación, y en pantalla se indica que se trata de la distancia entre ángulos medida en radianes. Esto se observa en la figura 54.

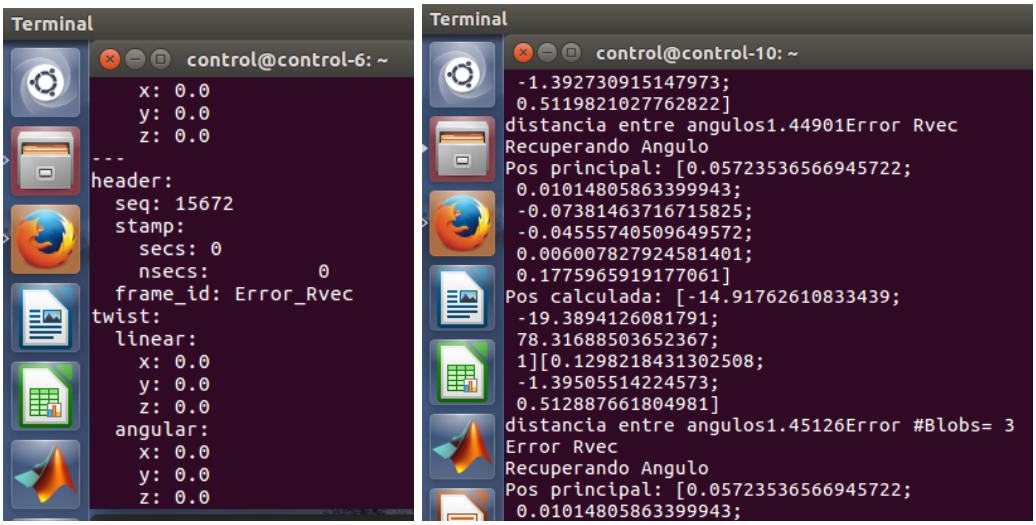

Figura 54 Recuperación de la información

6.7 Pruebas de Control

Como se explicó anteriormente en este capítulo, es necesario conocer el valor del potenciómetro del canal de Empuje, thrust, en el Joystick. Esto se puede observar en la figura 55 en donde se muestra como se conserva este valor en el modelo después de ser leído.

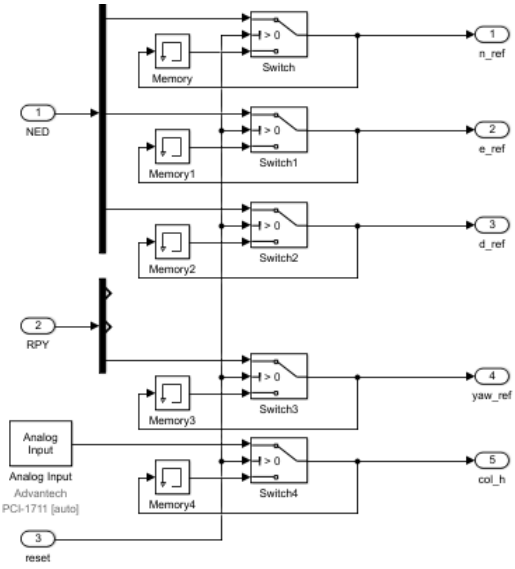

Figura 55 Entrada Analógica al sistema

De la misma forma se muestra en la figura 56 se observa como los valores calculados de acciones de control son escaladas para corresponderse con los voltajes del helicóptero y se establecen límites mediante saturaciones antes de que el dato sea escrito en la salida digital correspondiente.

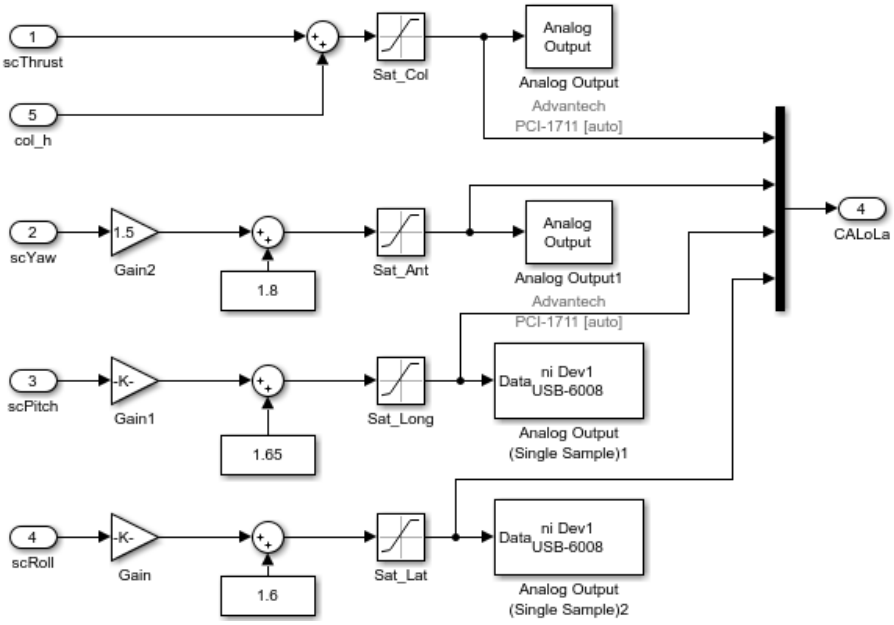

Figura 56 Salidas Analógicas del sistema

En la tabla 6 se observa los diferentes valores límites que se usaron para calcular el valor de compensación que se debe añadir a cada acción de control.

Tabla 6 Valores de tensión de cada canal en el control remoto

	Mínimo [V]	Centro [V]	Máximo [V]
Colectivo	0,38		3,12
Antipar	0,29	1,74	3,34
Eje Lateral	0,27	1,62	2,98
Eje Longitudinal	0,29	1,66	2,93

Con estos esquemas se ha comprobado, con el control remoto encendido, que no existen problemas de comunicación entre este y las tarjetas de entradas y salidas analógicas utilizadas, ya que cuando el control remoto se encuentra apagado sí se produce un error, debido a que en Matlab existe un número máximo de valores que no se pudieron escribir en una salida digital.

Para poder realizar una prueba de control del helicóptero desde Matlab se va a modificar el esquema de control anterior, de acuerdo al esquema que se observa en la figura 57.

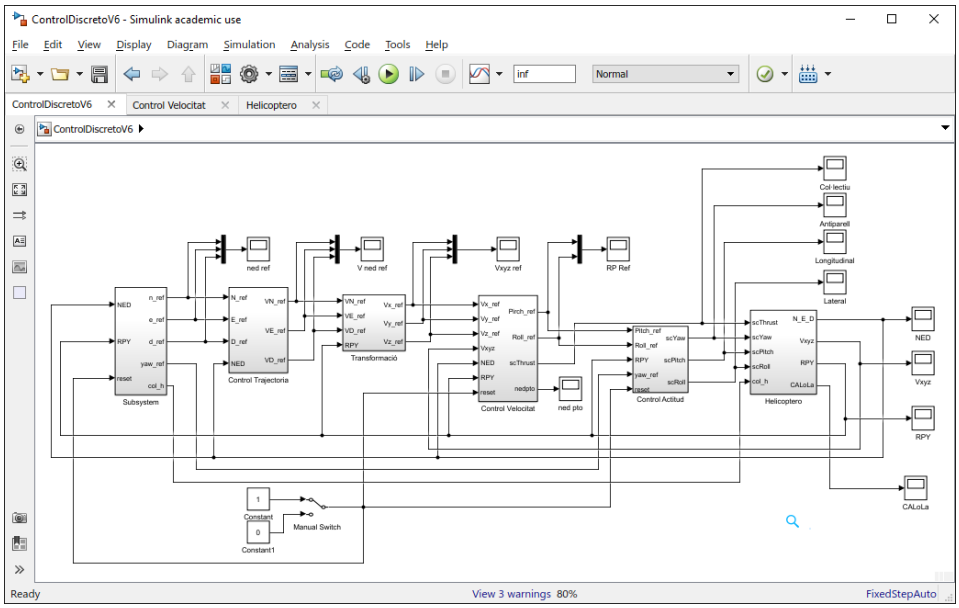

Figura 57 Esquema de Control Final

En este esquema se puede observar como primer punto que se ha substituido el bloque del helicóptero, en el cual antes se utilizaba un modelo lineal o no lineal del helicóptero por un bloque llamado helicóptero dentro del cual se ha programado el esquema de la figura 58.

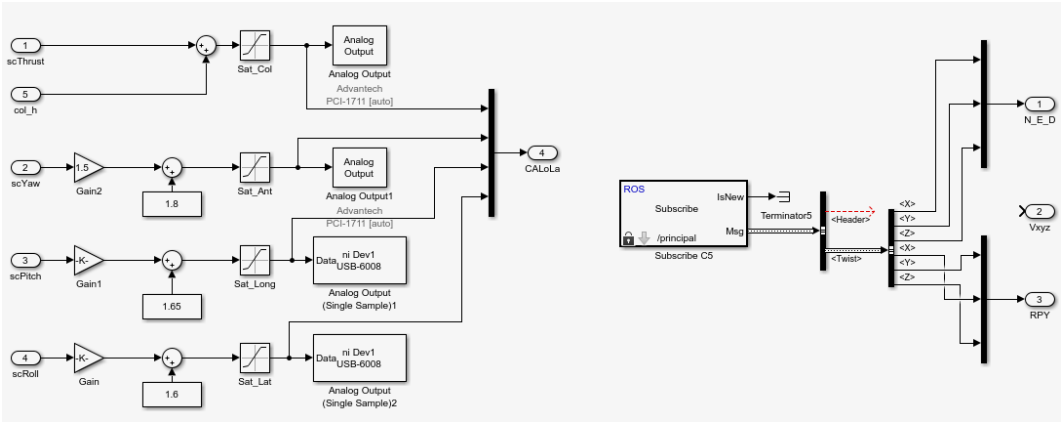

Figura 58 Bloque Helicóptero

Dentro de este bloque además de la conexión escalada de las salidas digitales, debe estar también la lectura del nodo principal, el cual tiene la información validada de la posición y orientación del helicóptero, junto con la correspondiente separación en los diferentes canales de posición y de orientación.

Adicionalmente se han cambiado los bloques PID dentro de los diferentes lazos, por unos bloques PID con un reset externo que se dispara con cualquier cambio de flanco. Este reset se ha implementado debido a que cuando el modelo comienza a funcionar, empieza a calcular un error que se acumula desde antes de iniciar el experimento, y debe ser puesto a cero, para que la parte integral no se encuentre saturada desde el inicio del experimento. Esta acción de reset se realiza con un switch manual al que se debe dar click cuando se desee iniciar el experimento.

Se ha programado también dentro del bloque referencias, el esquema de la figura 55 en donde se observa cómo se conserva los últimos valores de posición y del empuje leído, al momento de cambiar de estado el switch manual. Y finalmente se ha añadido una salida adicional al helicóptero para poder observar las acciones de control escaladas que se envían al helicóptero para que este mantenga el hover que se ha alcanzado al momento del experimento.

Con todas estas modificaciones se han realizado pequeñas pruebas para mantener el hover controlando el helicóptero desde Matlab. En estas pruebas se ha observado que aún se puede mejorar la sintonía de los diferentes lazos, o cambiar el tipo de control por técnicas más modernas.

7 CONCLUSIONES Y RECOMENDACIONES

El presente proyecto tenía como objetivo integrar diferentes sistemas, desarrollados por separado, para que trabajen conjuntamente de una manera adecuada y se ha conseguido satisfactoriamente.

Partiendo de un conocimiento nulo y ninguna experiencia sobre la plataforma ROS, se ha logrado comprender totalmente el funcionamiento de esta, y como se pueden utilizar, modificar y crear nuevos programas.

Uno de los problemas en el proyecto fue que la documentación referente al funcionamiento de ROS, está pensada y explicada para personas con un conocimiento previo de informática, especialmente en lo que se refiere al uso del sistema operativo Ubuntu Linux, es decir para usuarios de nivel intermedio de Linux.

También se ha logrado comprender el funcionamiento interno de un programa desarrollado por otra persona, en base a algoritmos de visión artificial disponibles en código abierto. De esta forma se ha conseguido actualizar los programas desarrollados a las versiones estables más recientes.

Se han realizado diferentes programas en código c ++, para mejorar el sistema de visión artificial, y hacerlo de esta forma más robusto a los cambios del entorno. Estos cambios pueden ser de tipo externos, como cambios en la luminosidad del ambiente, o internos como errores en el orden de los vectores necesarios para realizar los cálculos.

Gracias a este proyecto se ha aprendido también, cómo se realizan las comunicaciones a través de ROS, especialmente en la configuración de publicador subscriptor y como se puede aprovechar esta arquitectura para tener una red de fácil acceso a diferentes usuarios, o nodos.

A través de este trabajo se ha aprendido mucho sobre el funcionamiento de los helicópteros, especialmente sobre el helicóptero coaxial, y los diferentes elementos necesarios para su control. Un elemento que puede afectar el control del mismo es un giroscopio, el cual, si no se instala debidamente, es incapaz de ayudar a ajustar las velocidades de los motores e igualar sus velocidades.

Un problema muy difícil de resolver ha sido al peso que el helicóptero puede cargar, ya que los motores tienen una velocidad máxima, que determina una fuerza de sustentación máxima. Si se aumenta la carga entonces la fuerza de sustentación no es capaz de elevar el helicóptero.

Las baterías juegan también un papel importante en este tema, ya que determinan la potencia que el motor podrá desarrollar, buscando siempre que estas permitan obtener la potencia máxima de los motores, en vez de limitarlo en un rendimiento menor.

Adicionalmente se ha aprendido sobre el toolbox de Matlab “robotic system toolbox” y las diferentes herramientas de este, las cuales permitirán desarrollar proyectos grandes y pequeños.

El control del helicóptero se ha realizado utilizando múltiples lazos anidados, debido a que se trata de un sistema muy acoplado, y sobre todo muy inestable, por lo que no es capaz de soportar grandes perturbaciones.

Finalmente se ha observado que el sistema de visión ha conseguido un buen resultado siguiendo el movimiento del helicóptero en vuelo y las acciones de control calculadas son coherentes con el comportamiento del sistema.

8 BIBLIOGRAFÍA

[1]http://esaiict.upc.edu/microUAV/index.php/Objectives_and_description

[2]http://www.ros.org/

[3]http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

[4]http://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html

[5]https://es.mathworks.com/help/robotics/examples/get-started-with-ros.html

[6]https://es.mathworks.com/help/robotics/examples/get-started-with-ros-in-simulink.html

[7]http://www.rcmodels.com.ua/files/z55-1024g0103_expe.pdf.

[8]http://wiki.ros.org/ROS/Tutorials/